Zentralinstitut für Engineering, Elektronik und
Analytik (ZEA) · Systeme der Elektronik (ZEA-2)

Konzeptionierung einer Datenmanagementsoftware für die Verwaltung des Laborequipments

Klara Schnorrenberg

Jül-4436

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
Forschungszentrum

Zentralinstitut für Engineering, Elektronik und
Analytik (ZEA) · Systeme der Elektronik (ZEA-2)

Konzeptionierung einer Datenmanagementsoftware für die Verwaltung des Laborequipments

Klara Schnorrenberg

Berichte des Forschungszentrums Jülich
Jül-4436 · ISSN 0944-2952
Zentralinstitut für Engineering,
Elektronik und Analytik (ZEA)
Systeme der Elektronik (ZEA-2)
82 (Bachelor FH Aachen, Campus Jülich, 2022)

Vollständig frei verfügbar über das Publikations-
portal des Forschungszentrums Jülich (JuSER)
unter www.fz-juelich.de/zb/openaccess

Forschungszentrum Jülich GmbH · 52425 Jülich
Zentralbibliothek, Verlag
Tel.: 02461 61-5220 · Fax: 02461 61-6103
zb-publikation@fz-juelich.de
www.fz-juelich.de/zb

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.
Ich verpflichte mich, ein Exemplar der Bachelorarbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Ort, Datum	Klara Schnorrenberg

Kurzfassung

Die Informationen des Laborequipments werden im ZEA-2 aktuell nicht zentral gespeichert. Die Daten werden entweder analog an verschiedenen Orten des Instituts abgeheftet oder werden über die Qualitätsmanagementsoftware *Consense* verwaltet. Unter anderem damit alle Daten zentral abgespeichert werden und verwaltet werden können, ist das Ziel dies über eine Software zu implementieren.

Die vorliegende Arbeit beschäftigt sich mit der Konzeptionierung einer bilingualen *Datenmanagementsoftware*. Diese soll die Daten benutzerfreundlich darstellen und dem Anwender zielgerichtete Informationen liefern. Die Software wird von den Labormitarbeitenden und dem Qualitätsmanagementteam verwendet werden. Dabei wird erläutert, welche Anforderungen an die Software die beiden Gruppen haben.
Zudem wird auf die Anforderungen der Funktionalität eingegangen. Eine dieser Funktionalitäten ist, dass Geräte durch einen QR-Code einlesbar sein sollen. Damit können die Daten schnell visualisiert werden oder ein Messsetup benutzerfreundlich erstellt werden, indem alle gewünschten Geräte eingelesen werden.
Bei der Konzeptionierung dieser Software wird analysiert, wie die Daten am besten abgespeichert und und in welcher Form angezeigt werden. Als Datenbank wird sich für *MongoDB* entschieden.
Da es keine Vorgaben dazu gibt, ob die Software als eine Desktop- oder Webanwendung umgesetzt werden soll, wird auf eine Unterscheidung dieser eingegangen. Nach der Entscheidung für eine Webanwendung wird der *FARM*- und der *MERN*-Stack vorgestellt.
In der Realisierung wird dann anhand des FARM-Stacks ein „Proof of Concept“ erarbeitet. Dabei wird auf die Abspeicherung der Daten in *MongoDB* eingegangen und die Kommunikation von der Weboberfläche zu der Datenbank beschrieben.
Zuletzt werden die Umsetzungen der Bilingualität und das Erstellen der Qr-Codes mittels einer generierten Excel-Datei vorgestellt.

Aus Gründen der besseren Lesbarkeit wird auf die gleichzeitige Verwendung der Sprachformen männlich, weiblich und divers (m/w/d) in dieser Arbeit verzichtet. Sämtliche Personenbezeichnungen gelten gleichermaßen für alle Geschlechter.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Dieses Kapitel stellt zunächst im Unterkapitel 1.1 das Zentralinstitut für Engineering, Elektronik und Analytik vor. Dabei wird insbesondere auf den Institutsbereich des *ZEA-2* eingegangen, in welchem diese Arbeit verfasst wurde. Anschließend folgt im Unterkapitel 1.2 die Motivation dieser Arbeit.

1.1 Zentralinstitut für Engineering, Elektronik und Analytik

Das Zentralinstitut für Engineering, Elektronik und Analytik (ZEA) ist Teil der Forschungszentrum Jülich GmbH. Der Fokus der Forschung liegt auf der Entwicklung von Geräten, Experimenten, Prozessen, Analyseverfahren, Mess-, Analyse- und Regelungsanlagen, Detektorsystemen und computergestützten Werkzeugen sowie bildgebenden Verfahren, die es nicht auf dem Markt zu kaufen gibt. Diese Forschung ist eng an die anderen Jülicher Institute, sowie an Universitäten und wissenschaftliche Einrichtungen weltweit verbunden.[1]

Das Zentralinstitut besteht aus drei Institutsbereichen:

1. Engineering und Technologie (ZEA-1)
2. Systeme der Elektronik (ZEA-2)
3. Analytik (ZEA-3)

Die Arbeit wurde im Institutsbereich Systeme der Elektronik erstellt, weshalb auf diesen im folgenden Abschnitt kurz eingegangen wird.

Systeme der Elektronik (ZEA-2)

Das ZEA-2 ist auf die Entwicklung komplexer elektronischer und informationstechnischer Systeme für die Forschung spezialisiert. Dabei ist das Leitbild des Instituts, sowohl Spitzenforschung als auch Innovationen für die Gesellschaft zu ermöglichen. Dies wird durch elektronische Systemlösungen, vorzugsweise durch integrierte Schaltkreise und unter Verwendung von modellbasierten Systemanalysen und wissenschaftlichen Methoden erzielt. Die wissenschaftlichen Schwerpunkte liegen auf Mess- und Detektorsystemen, sowie elektronischen Systemen für Quantum und Neuromorphic Computing.[2]

1.2 Motivation

Das Kapitel der Motivation ist in drei Abschnitte aufgeteilt. Im ersten Abschnitt 1.2.1 wird das bisherige Vorgehen der Datenverwaltung erläutert. Aufbauend auf diesem Abschnitt folgen im nächsten Abschnitt 1.2.2 mögliche Optimierungen des bisherigen Vorgehens. Im letzten Abschnitt 1.2.3 werden die Ziele einer *Datenmanagementsoftware* dargestellt.

1.2.1 Bisheriges Vorgehen

Bisher werden die Informationen des Laborequipments weitestgehend analog an verschiedenen Orten im Institut in Ordnern abgeheftet. Dies erschwert das gezielte Nachschlagen von bestimmten Informationen, da erst einmal der richtige Ordner gefunden werden muss und dann die richtige Seite mit der gesuchten Information des Messgeräts. Dies ist einerseits nicht zeiteffizient und andererseits kann es durch zum Beispiel falsches Abheften auch zu Datenverlust kommen. Des Weiteren sind die Informationen nur verfügbar, wenn sich die Mitarbeiter vor Ort befinden.
Manche Informationen werden bereits digital mit einer Software namens *ConSense* verwaltet. Dies geschieht vor dem Hintergrund, dass das *ZEA-2* ISO-zertifiziert ist und hierfür gewisse ISO-Normen erfüllen muss. Zur Erfüllung eines Teils dieser Vorgaben, wird derzeit im *ZEA-2 ConSense* verwendet.

ConSense ist eine Software für das Prozessmanagement, Qualitätsmanagement und die integrierten Managementsysteme [3]. In vielen Unternehmen müssen Anforderungen an die Prozesse und deren Dokumentationen eingehalten werden. Diese Anforderungen sind in sogenannten Normen festgelegt. Für viele Unternehmen gilt die DIN EN ISO 9001 Norm.

Das DIN steht dabei für das „Deutsche Institut für Normung“, das „EN“ steht für die „Europäische Norm“ und das ISO ist die Abkürzung für „International Organization for Standardization“. Eine Norm, die alle drei Begriffe beinhaltet, wird weltweit anerkannt. Mittels dieser Software können entsprechende Dateien abgelegt und bestimmte Informationen der Geräte gespeichert werden.

Im *ZEA-2* wird *ConSense* verwendet, um die zu den Geräten notwendigen Dokumente zu speichern. Hierzu zählen beispielsweise die Kalibrierungsberichte der Laborgeräte, welche in den Audits, bei denen die Zertifizierung überprüft wird, vorzeigbar sein müssen.

1.2.2 Optimierungsmöglichkeiten des bisherigen Vorgehens

Da die Daten im aktuellen Vorgehen nicht gebündelt sind, sondern aus verschiedenen Quellen gezogen werden müssen, wäre es sinnvoll, diese durch eine Software zusammenzufassen.
Damit soll außerdem die Verwendung von *ConSense* im *ZEA-2* abgeschafft werden, da die spezifische Benutzung für das Institut nicht vollständig zufriedenstellend ist. Für die Änderung von Daten gibt es bestimmte Genehmigungsprozesse, in denen Freischaltungen vorgenommen werden müssen. Bei der Größe des Forschungszentrums können diese Prozesse und Freischaltungen sehr zeitaufwendig sein. Dieses erschwert ebenfalls die Wartbarkeit und die Erweiterbarkeit.
Die Vorteile der Digitalisierung der Daten liegen darin, dass die Daten an einem zentralen Ort liegen und sich der Zugriff für die Mitarbeiter vereinfachen kann. Das Ablegen der Daten kann sich beschleunigen und physischer Speicherplatz wird eingespart, da nicht alles in Akten aufbewahrt werden muss.

Außerdem gibt es weitere interne Anforderungen, die den Laborbetrieb deutlich optimieren sollen. Zum Beispiel ist ein weiterer Aspekt dabei, dass die spätere Messsoftware erkennen soll, ob die gewählte Hardware überhaupt nutzbar ist. Nur Hardware, die kalibriert und nicht in einem anderen Setup genutzt wird, kann verwendet werden. Derzeit können diese Informationen nirgends vorgehalten werden, sondern müssen für jede Messung manuell geprüft werden.

1.2.3 Ziel einer *Datenmanagementsoftware*

Ziel dieser Arbeit ist es, ein Konzept für eine *Datenmanagementsoftware* zu entwickeln, die den Laborbetrieb verbessert. Dazu soll die Software einerseits die Informationen des Laborequipments verwalten können und somit alle Daten zentral speichern. Andererseits sollen mit Hilfe der Software gezielt die verfügbaren Informationen für

den Arbeitsalltag benutzt werden können, zum Beispiel vor oder nach Messungen. Um eine einfache Verwaltung des Laborequipments zu gewährleisten, ist ein weiteres Ziel der *Datenmanagementsoftware* die Daten benutzerfreundlich darzustellen. Dabei sollen für den Anwender zielgerichtete Informationen dargestellt werden. Im *ZEA-2* gibt es verschiedene Anwendungsbereiche, für die die Software interessant ist, weshalb nach Stakeholdern unterschieden wird. Im Unterkapitel 2.2 wird auf diese näher eingegangen.

Die genauen Anforderungen und Zielsetzungen werden im weiteren Verlauf beschrieben.

2 Anforderungen und Zielsetzungen

In diesem Kapitel werden die Anforderungen an die *Datenmanagementsoftware* definiert und die daran gesetzten Ziele vorgestellt. Zu Beginn des Unterkapitels 2.1 wird auf das Gesamtprojekt eingegangen. Dabei werden im Unterkapitel 2.2 die verschiedenen Stakeholder für die verschiedenen Anwendungsbereiche der Software vorgestellt. In dem darauf folgenden Unterkapitel 2.3 werden die geplanten Funktionalitäten der Software erläutert.

2.1 Vorstellung des Gesamtprojekts

Die Anforderungen an das Projekt, dessen Ziel die Optimierung des Laboralltags ist, bestehen aus zwei Bereichen. Dieses ist schematisch in der Abbildung 2.1 dargestellt. Dabei deckt diese Arbeit den linken Teil ab und wird im weiteren Verlauf beschrieben. Der Teil rechts von der Datenbank wird in einer weiteren Bachelorarbeit von Daniel Keßel [1] thematisiert. Dabei geht es um eine Generalisierungen von SCPI-Befehlen. SCPI ist ein Befehlssatz zur Ansteuerung und Programmierung von Messgeräten. Damit typgleiche Messgeräte, die jedoch von anderen Anbietern sind, auf die gleiche Weise angesteuert werden können. Die Daten werden dabei in einer Datenbank gespeichert, wie auch in der Abbildung 2.1 verdeutlicht. Für die SCPI-Befehle sind auch entsprechende Geräteinformationen relevant, wodurch eine gemeinsame Datenbank speichereffizienter ist und es zu keiner inkonsistenten Datenhaltung durch mehrfaches Abspeichern der Daten in zwei Datenbanken kommen kann. Durch die gemeinsame Schnittstelle der Arbeiten wurde sich in der Entscheidung des Datenbankmanagementsystems abgestimmt, so dass die Anforderungen beider Teile an die Datenbank abgedeckt werden.

[1] Bachelorarbeit von Daniel Keßel unter dem Titel: „Konzeptionierung einer universellen Datenschnittstelle für die Ansteuerung von Labormessgeräten mittels SCPI“

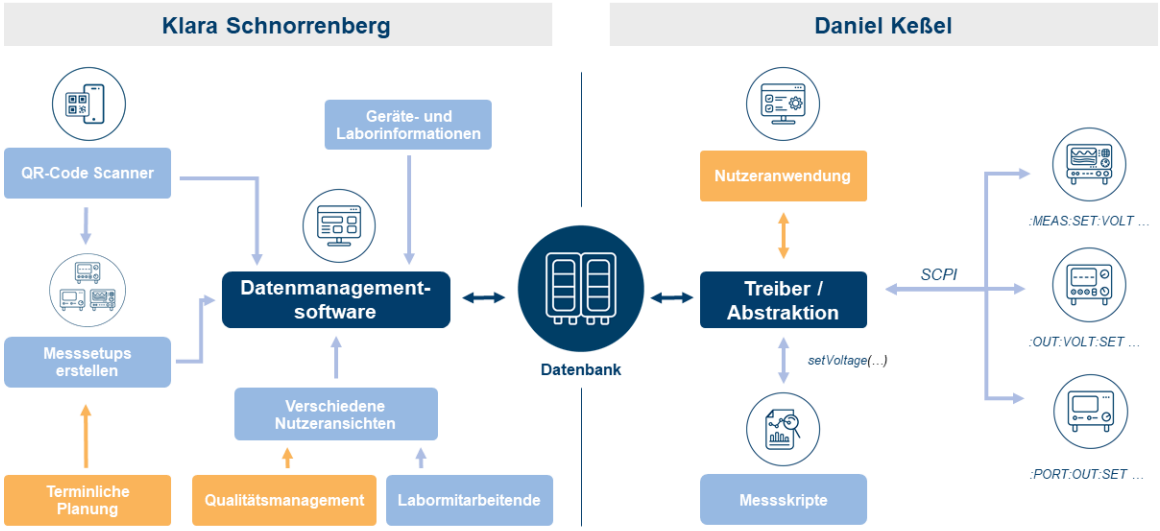

Abbildung 2.1: Darstellung des Gesamtprojekts

2.2 Stakeholder der *Datenmanagementsoftware*

Die Verwendung der neuen Software ist für verschiedene Anwendungsbereiche interessant und daher müssen auch die individuellen Anforderungen berücksichtigt werden. Derzeit bekannt ist, dass mindestens zwei Gruppen die Software verwenden sollen: die Labormitarbeiter und das Qualitätsmanagementteam. Auf diese Stakeholder wird näher eingegangen.

Die Gruppe der Labormitarbeiter lässt sich zusätzlich noch unterteilen in Laborverantwortliche, die für die Hardware verantwortlich sind, und die normalen Mitarbeitenden, die die Hardware meist für Messungen nutzen. In der täglichen Arbeit verwenden die Labormitarbeiter die Hardware für verschiedene Messungen. Hierbei ist eine sehr wichtige Information, ob ein Gerät kalibriert ist. Nur kalibrierte Messgeräte dürfen zur Verifikation genutzt werden, da ein nicht kalibriertes Gerät die Messwerte verfälschen kann. Innerhalb sogenannter Messsetups werden mehrere Geräte angeschlossen. Des Weiteren haben die Geräte verschiedene Optionen. Daher ist es notwendig zu wissen, welches Gerät welche Optionen bietet. Im realen Laborbetrieb kann es vorkommen, dass Messgeräte einfach abgebaut werden, um sie in einem anderen Setup zu nutzen. Daher ist der Laborverantwortliche dafür zuständig, dass die Geräte auffindbar und nutzbar sind.

Eine weitere Aufgabe des Laborverantwortlichen ist es, für die rechtzeitige Kalibrierung der Hardware zu sorgen. Im Falle von Problemen wäre eine übersichtliche Darstellung zu den Herstellern der Messgeräte hilfreich, um so eine schnellere Kontaktaufnahme zu ermöglichen.

Für die Anforderungen in dieser Arbeit wird jedoch auf die Unterscheidung zwischen Laborverantwortlichem und Mitarbeiter verzichtet, da die Aufgabenbereiche z.B. über die Vergabe von Rechten ermöglicht werden kann.

Ein weiterer Stakeholder ist das Qualitätsmanagementteam. Da das *ZEA-2* die ISO-Norm einhalten möchte, beschäftigt sich dieses Team damit, die Daten gemäß dieser Norm konsistent zu halten. In entsprechenden Audits wird dies auch regelmäßig geprüft. Dabei müssen die Dokumente, wie zum Beispiel die Kalibrierungsberichte, aber auch Messprotokolle, vorzeigbar sein. Ebenfalls kann es auch sein, dass bestimmte Dokumente vorgezeigt werden müssen, falls bei einem Gerät die Eingangsprüfung der Kalibrierung nicht erfolgreich verlief. In den Audits müssen bestimmte Prozesse nachvollziehbar sein, wie zum Beispiel der Ablauf einer Kalibrierung und ein womögliches Handeln bei einer nicht erfolgreichen Eingangsprüfung.

2.3 Funktionalitäten der Software

Neben dem Aspekt, dass die Software alle Informationen übersichtlich darstellen soll, gibt es weitere Funktionalitäten, die diese beinhalten soll. Auf die verschiedenen Funktionalitäten wird in den folgenden Abschnitten näher eingegangen.

2.3.1 Anmeldeverfahren mit spezifischen Rechten

Zur Nutzung der eigentlichen Software ist ein Anmeldeprozess vorgesehen. Dies ermöglicht es, dass je nach Benutzer verschiedene Rechte zugeordnet werden können. Das ist wichtig, da die Laborverantwortlichen mehr Rechte haben als die restlichen Mitarbeitenden. Hierdurch kann die Unterscheidung der Labormitarbeiter als Stakeholder berücksichtigt werden. Außerdem soll durch eine Anmeldung eine Rückverfolgung möglich sein, wer bestimmte Daten zuletzt geändert hat. Dies wiederum ist für den Bereich des Qualitätsmanagements wichtig.

Dabei gibt es drei Formen von Rechten:

1. „nur lesenden“ Zugriff: Möglich für alle Mitarbeiter und Benutzer (ohne Anmeldung möglich)

2. Admin-Rechte: Uneingeschränkter Lese- und Schreibzugriff

3. eingeschränkte Rechte: Werden für die Benutzergruppe individuell angepasst

Geplant ist es, dass für die Anmeldung in der Software die LDAP-Daten benutzt werden. LDAP-Daten sind zentral genutzte Anmeldedaten. Dies hat den Vorteil für die Benutzer, dass sie sich keine zusätzlichen Anmeldeinformationen merken müssen, und für die Administration, dass keine zusätzliche Benutzerdatenverwaltung nötig ist.
Ein weiterer Vorteil eines Anmeldeverfahrens ist, dass auf den Benutzer zugeschnittene Informationen angezeigt werden können. Dies kann den Vorteil haben, dass die in Unterkapitel 2.2 auf Seite 6 beschriebenen Stakeholder, durch die Anmeldung definiert werden können. Dadurch kann gezielt auf den individuellen Benutzer eingegangen werden und damit entsprechende Ansichten generiert werden.

2.3.2 Messsetup

Im Unterkapitel 2.2 wurde bereits die Verwendung von Messsetups kurz erwähnt. Der Benutzer, in diesem Fall der Labormitarbeiter, soll die Möglichkeit haben, ausgewählte Geräte zusammenzufassen. Diese Zusammenfassung wird als Messsetup bezeichnet. Zur Unterscheidung mehrerer Messsetups werden diese mit einem Namen versehen. Des Weiteren soll somit auch ermöglicht werden, dass der Benutzer Informationen erhält, welche Geräte gerade in Benutzung sind. Aus diesen Messsetups lassen sich die relevanten Informationen für z. B. Messprotokolle extrahieren, die wiederum im Rahmen der ISO-Zertifizierung wichtig sind.
In einem späteren Entwicklungsschritt kann die Software noch entsprechend erweitert werden, um auch die zukünftige Planung der Geräte zu integrieren. Dadurch wäre es möglich, dass Geräte für einen festen Zeitraum und für ein bestimmtes Projekt gebucht werden können. Dies würde einen besseren Überblick geben, ob alle für ein Projekt benötigten Geräte in einem bestimmten Zeitraum verfügbar sind.

2.3.3 Verschiedene Ansichten der Benutzeroberfläche

Wie im Unterkapitel 2.2 auf Seite 6 beschrieben, gibt es verschiedene Stakeholder, die an der Nutzung der Software interessiert sind. Da die Faktoren je nach Stakeholder unterschiedlich sind, soll mittels verschiedener Benutzeroberflächen eine für den Stakeholder spezifische Anzeige ermöglicht werden.
Für das Qualitätsmanagementteam ist zum Beispiel das Einhalten und Nachvollziehen bestimmter Prozesse anhand von Dateien wichtiger als zu wissen, welche Optionen ein Gerät besitzt. Dies ist für die Labormitarbeitenden jedoch genau andersherum. Eine Unterscheidung der Benutzeroberfläche erzielt somit eine auf die Anwendung individuell zugeschnitten übersichtliche Darstellung der Daten.

2.3.4 Integrierte Benutzung von QR-Codes

Es soll eine Möglichkeit bereitgestellt werden, ohne Anmeldung ein Gerät mittels eines QR-Scanners einzuscannen und dann entsprechende Daten des Geräts angezeigt zu bekommen. Hierbei handelt es sich um einen reinen Lesezugriff. Dies bedeutet, dass für eine Änderung dieser Daten dann jedoch eine Anmeldung vorausgesetzt wird.
Der QR-Code soll außerdem verwendet werden können, um Geräte einzuscannen. Dies kann zum Beispiel bei der Auswahl der Geräte für ein Messsetup Anwendung finden. Das ermöglicht, dass der Benutzer im Labor die spezifischen Geräte einscannen kann, ohne dass er das Gerät über eine Liste oder durch spezielle Angaben zu dem Gerät suchen muss. Eine solche Auswahl der Hardware ist auch im Zusammenhang mit der Kalibrierung sinnvoll. So können mittels der QR-Codes alle Geräte in einem Labor gescannt werden, die für eine Kalibrierung vorgesehen werden müssen.

2.3.5 Erzeugen der QR-Codes

Um die erwähnten QR-Codes nutzen zu können, sollen diese auf den Laborgeräten angebracht werden. Um diese Codes auszudrucken, wird ein entsprechender QR-Code-Drucker der Firma *Brother* verwendet. Zur Erzeugung und für den anschließenden Druck wird ein mitgeliefertes Herstellertool verwendet. Dieses Tool kann mit Hilfe einer Excelliste gesteuert werden. Daher soll es eine Möglichkeit geben, eine Excel-Datei zu erstellen, die entsprechenden Daten aus der Software zu extrahieren und den QR-Code damit zu generieren. Diese Datei enthält die benötigten Daten für alle ausgewählten Geräte. Die Excel-Datei muss dabei nach einem festen Format aufgebaut sein. In dem Tool von *Brother* kann dann die Excel-Datei eingelesen und die gewünschten QR-Codes erzeugt und ausgedruckt werden.

2.3.6 Bilinguale Benutzeroberfläche

Das *ZEA-2* ist ein multinationales Institut. Die Mitarbeitenden sprechen dabei alle mindestens entweder Deutsch oder Englisch. Deshalb soll bei der Konzeptionierung berücksichtigt werden, dass die Software eine bilinguale Benutzeroberfläche besitzen sollte. Dadurch ist die Software universell für alle nutzbar und jeder Nutzer kann entscheiden, ob die Komponenten in der Benutzeroberfläche in deutscher oder englischer Sprache anzeigt werden sollen. Die Anpassung der Sprache wirkt sich nur auf die graphischen Elemente in der Oberfläche aus, jedoch nicht auf die in der Datenbank hinterlegten Informationen oder Nutzereingaben.
Somit sollte institutsintern ein gemeinsames Vorgehen der Nutzer bei der Eintragung angestrebt werden, sodass auch die abgespeicherten Informationen einheitlich in einer Sprache abgelegt werden. Hierfür bietet sich Englisch an, da dies in der Wissenschaft die führende Sprache ist. Innerhalb des Instituts wurde ein solcher Vorschlag einer einheitlichen Schriftsprache bereits begonnen. Um die Akzeptanz des neuen Tools zu erhöhen, soll trotz dieses Prozesses eine bilinguale Benutzeroberfläche ermöglicht werden.

2.4 Zielsetzung

Aus den zuvor beschriebenen Anforderungen soll im Rahmen dieser Bachelorarbeit ein Konzept einer *Datenmanagementsoftware* für die Verwaltung des Laborequipments entwickelt werden.
Dazu werden die wesentlichen Aspekte der Aufgabe noch einmal zusammengefasst. Die geplante bilinguale Software soll dabei eine Schnittstelle zu einer Datenbank besitzen. Um die Struktur der Datenbank zum Abspeichern der relevanten Hardwareinformationen erstellen zu können, müssen die Geräteinformationen zunächst analysiert werden. Dabei ist neben der Datenhaltung auch die spätere Visualisierung zu entwerfen. Dazu soll geprüft werden, ob sich für diesen Anwendungsfall besser eine Webanwendung oder eine Desktopanwendung eignet.
Im ersten Schritt wird dabei vor allem auf die Bedürfnisse der Labormitarbeiter eingegangen. Dennoch muss die Software so modular aufgebaut werden, dass sie ohne Probleme auch für weitere Stakeholder, wie zum Beispiel das Qualitätsmanagementteam, erweitert werden kann. Für die Labormitarbeiter werden die bereits beschriebenen Funktionalitäten berücksichtigt.
Ein weiterer wichtiger Punkt ist außerdem die Vorbereitung der Erstellung der QR-Codes. Neben dem Entwurf der Oberflächen und der QR-Codes muss auch die Kommunikation zwischen Datenbank und Anwendung betrachtet werden, da diese einen wichtigen Bestandteil des Konzepts darstellt.
Die Grundidee dieser Software ist in Abbildung 2.1 auf Seite 6 links skizziert.

3 Analyse von den Informationen des Laborequipments

Dieses Kapitel beschäftigt sich mit der Analyse der Informationen. Und behandelt im Unterkapitel 3.1 welche Kategorien von Informationen betrachtet werden. Da die Kalibrierung der Geräte eine wichtige Rolle spielt, wird im Unterkapitel 3.2 auf den Ablauf dieser eingegangen.

3.1 Kategorien der Informationen

Die Informationen des Laborequipments wurden in verschiedene Bereiche aufgeteilt. Zum einen gibt es die Geräteinformationen, die alle Daten eines Geräts zusammenfassen, zum anderen werden aber auch Daten von Mitarbeitern, Räumen oder Firmen betrachtet, da diese ebenfalls für die Arbeit mit den Geräten relevant sein können.
Im Folgenden werden alle Kategorien aufgelistet, von denen Informationen betrachtet werden sollen:

- Geräte
- Kalibrierung
- Messsetups
- Firmen
- Mitarbeiter
- Wartung
- Verleih von Geräten
- Verwertung der Geräte
- Reparatur der Geräte
- Labordaten
- Geräteoptionen

Ein Schwerpunkt wird in dieser Arbeit auf die Kalibrierungsinformationen gelegt, da diese für die Labormitarbeitenden als auch für das Qualitätsmanagementteam eine wichtige Rolle spielen.
Die Kategorien, die in dieser Arbeit fokussiert betrachtet werden, sind dabei die Informationen der Geräte, der Kalibrierungen und der Messsetups. Im Folgenden werden daher für diese Kategorien die dazugehörenden Angaben erläutert.

3.1.1 Geräteinformationen

Neben den Stammdaten eines Gerätes, wozu zum Beispiel die Seriennummer, Inventarnummer oder auch die Spezifizierung des Gerätes gehören, gibt es weitere Aspekte, die relevante Informationen eines Gerätes sind. Dazu zählen Kalibrierungsinformationen zu dem Gerät und des Weiteren mögliche Informationen zu dem Verleih, der Verwertung oder dem Verkauf des Gerätes.
Im Folgenden sind alle Eigenschaften, die zu einem Gerät gehören, aufgelistet. Hiermit soll der Umfang der zu speichernden Informationen verdeutlicht werden. Denn wie zu erkennen ist, gibt es sehr viele Informationen, die zu einem Gerät benötigt werden. Bei der weiteren Analyse werden aus Gründen der Übersichtlichkeit hauptsächlich die relevantesten Informationen verwendet.

Geräteinformationen

- Gerätebeschreibung
  - Bezeichnung
  - Modell
  - Hersteller
  - Gerätegruppe
  - Masse
  - Kalibrierungsintervall laut Hersteller
  - Kalibrierungsintervall im ZEA-2
  - Kalibrierungskosten
  - Wartungsintervall
- Anschaffungsdatum
- Seriennummer
- Inventarnummer des FZJ
- Inventarnummer des ZEA-2
- IP-Adresse
- USB-ID
- Lieferscheinnummer
- Labor
- Raum
- SAP-Nummer
- Einkaufsnettopreis
- Spezifizierung des Gerätes
- Kalibrierungsinformationen
  - Kalibrierung notwendig?
  - Kalibrierungsintervall
  - Nächstes Kalibrierungsdatum
  - Kalibrierungsfirma
- Verleih des Gerätes
  - Ausleihender
  - Verleihdatum
  - Rückgabedatum
- Aussortierung des Gerätes
  - Dokument der Aussortierung
  - Aussortierungsdatum
- Verkauf des Gerätes
  - Daten zum Käufer
  - Verkaufsdatum
  - Dokument des Verkaufs

Wie im Unterkapitel 2.1 auf Seite 5 beschrieben wurde, besteht das Gesamtprojekt momentan aus zwei Teilen. Es werden daher auch Informationen gespeichert, die vor allem für den Hardware-Ansteuerungsteil der Anwendung relevant sind. Diese Informationen sind hauptsächlich für die Verbindung mit den Geräten in der Daten-

schnittstelle relevant. Jedes Gerät muss dabei einzeln ansteuerbar sein und es soll aus unterschiedlichen Verbindungsmodi, wie zum Beispiel TCP/IP und USB, ausgewählt werden können. Daher müssen die Informationen für die Verbindung mit den Geräten, wie zum Beispiel die IP-Adresse oder die USB-ID, ebenfalls abgespeichert werden.

3.1.2 Messsetup

Um ein Messsetup zu erstellen, werden bestimmte Daten abgespeichert, damit ersichtlich wird, von wem das Messsetup erstellt wurde und wie lange die Geräte in Verwendung sein werden. Diese Informationen geben Auskunft darüber, ob ein Gerät momentan in einer geplanten Verwendung ist oder nicht.
Im folgenden Werden die Eigenschaften für ein Messsetup aufgelistet.

Daten für ein Messsetup:

- Name des Messsetups
- Verantwortlicher
- Labor
- Start- und Enddatum
- Liste an Geräten

3.1.3 Kalibrierungsdaten

Die meisten Geräte werden regelmäßig kalibriert. Dazu müssen entsprechende Daten dieser Kalibrierung abgespeichert werden, um eine Historie dieser Daten ermöglichen zu können.

Kalibrierungsdaten

- Informationen zur Servicefirma
- Kalibrierungsdatum
- Kalibrierung erfolgreich
- Eingangsprüfung erfolgreich
- Prüfprotokoll
- Rundschreiben notwendig
- Einbuchungsdatum

Die Servicefirma besteht dabei aus weiteren Eigentschaften:

- Firmenname
- Adresse
  - Straße
  - Hausnummer
  - Postleitzahl
  - Stadt
- Kundennummer
- Website
- Hinzugefügt am
- Hinzugefügt von
- Kontaktperson
  - Titel
  - Vorname
  - Nachname
  - E-Mail
  - Telefon
  - Ist Hauptansprechspartner?
- Kürzel
- Ist gesperrt?

Um den Prozess einer Kalibrierung nachzuvollziehen, wird im nächsten Unterkapitel auf diesen Vorgang genauer eingegangen.

3.2 Ablauf der Geräte-Kalibrierung

Die meisten Geräte müssen regelmäßig kalibriert werden. In der Abbildung 3.1 auf Seite 15 ist der Ablauf einer Kalibrierung dargestellt. Nachdem der Benutzer über die Oberfläche die zu kalibrierenden Geräte ausgewählt hat, werden diese in einer Datei gelistet. Zeitgleich soll dabei ein neuer Kalibrierungsantrag in der Datenbank gespeichert werden. Es soll übersichtlich dargestellt werden, wann ein Antrag gestellt wurde, von wem und welche Geräte dabei kalibriert werden sollen.
Mit der erzeugten Datei, in der die Geräte gelistet sind, wird ein Kalibrierungsauftrag bei der entsprechenden Firma gestellt. Nachdem dieser Auftrag angenommen wurde, werden die Geräte zu den Firmen verschickt. In der Software soll dann der Benutzer dies bei dem Kalibrierungsantrag vermerken können, wodurch der Ort der Geräte sich auf „Außer Haus“ ändern soll.
In der Eingangsprüfung werden die Geräte mit genaueren Geräten verglichen, um festzustellen, ob diese noch den erwarteten Werten entsprechen. Falls die Eingangsprüfung nicht erfolgreich verlaufen ist, muss dies dem Mitarbeiter mitgeteilt werden. Denn es kann vorkommen, dass Messungen mit dem Gerät durchgeführt wurden. Da das Gerät nicht richtig kalibriert war, wurden falsche Werte angenommen und möglicherweise veröffentlicht. Momentan wird dies durch die Laborverantwortlichen kommuniziert, die dies an die entsprechenden Mitarbeiter weiterleiten. Ziel ist es, diesen Prozess zu vereinfachen, indem ein Rundschreiben per E-Mail verschickt wird mit allen Geräten, die bei der Eingangsprüfung nicht bestanden haben.

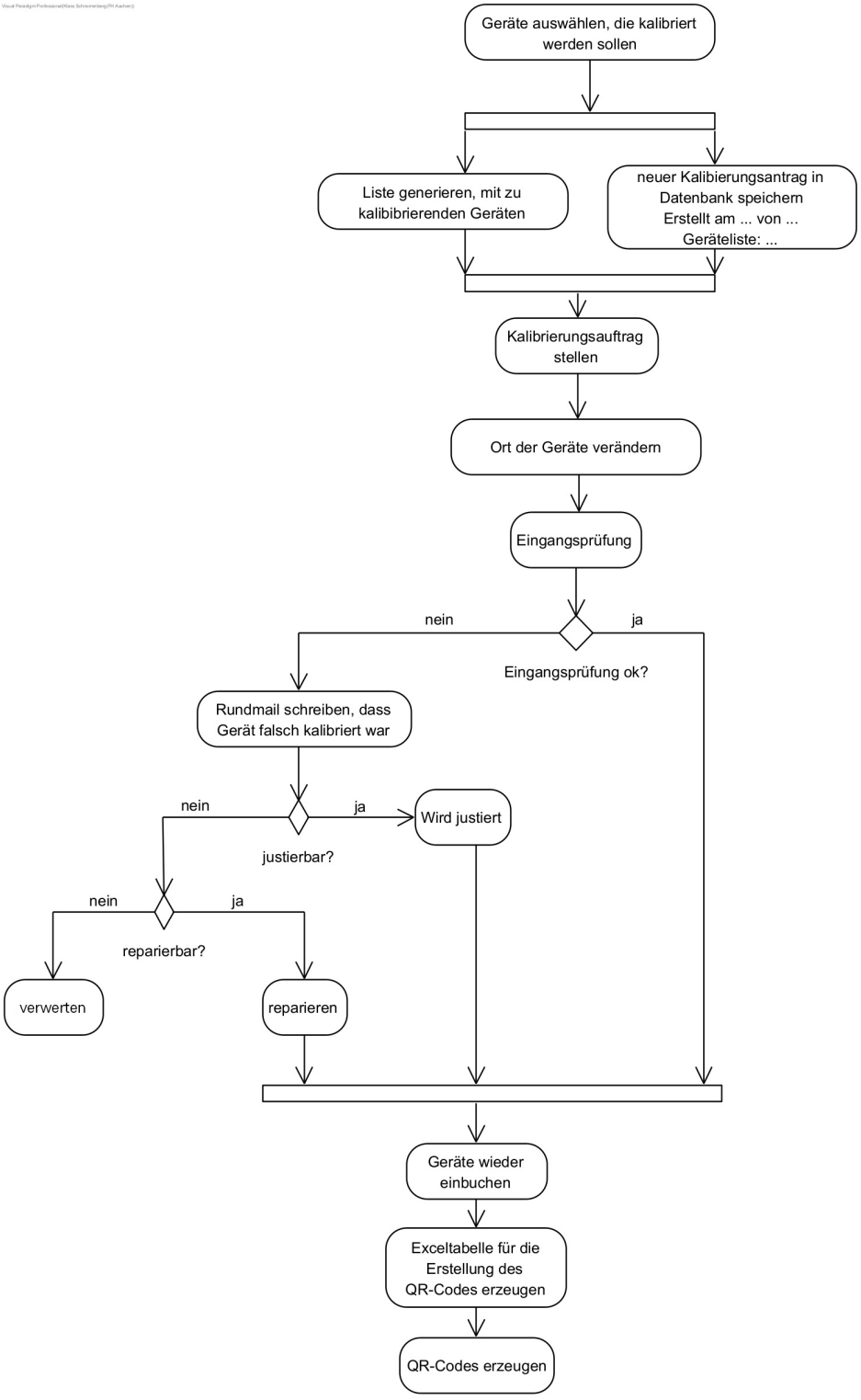

Abbildung 3.1: Aktivitätsdiagramm des Kalibrierungsablaufs

Anschließend werden die Geräte daraufhin überprüft, ob diese sich korrekt justieren lassen. Wenn dies nicht möglich ist, wird geprüft, ob diese repariert werden können. Fall eine Reparatur nicht möglich ist oder diese nicht wirtschaftlich ist, wird das Gerät aussortiert, ansonsten wird es repariert.
Alle Geräte, die kalibriert werden, sollen danach wieder eingebucht werden können. Dabei soll der Ort des Gerätes wieder angepasst werden. Zudem soll es die Möglichkeit geben, entsprechende Dokumente der Kalibrierung hochzuladen. Anschließend wird eine neue Exceltabelle mit den angepassten nächsten Kalibrierungsterminen der Geräte erzeugt. Wie in dem Abschnitt 2.3.5 auf Seite 9 beschrieben, wird diese verwendet, um die QR-Codes mit dem *Brother*-Drucker zu erstellen.

4 Konzeptionierung

In dem Kapitel der Konzeptionierung wird sich in dem ersten Unterkapitel 4.1 mit dem Datenbankkonzept beschäftigt. Dabei wird auf die Entscheidung für ein dokumentbasiertes Datenbanksystem eingegangen und anschließend *MongoDB* erklärt. Im darauf folgenden Unterkapitel wird das Konzept der Benutzeroberfläche thematisiert. Dazu wird zu erst auf den Entwurf der Benutzeroberflächen eingegangen. Anschließend folgt eine Unterscheidung zwischen Web- und Desktopanwendung. Abschließend wird in diesem Unterkapitel begründet wieso dieses Projekt über eine Webanwendung realisiert wird.

4.1 Datenbankkonzept

In diesem Unterkapitel wird das Datenbankkonzept erläutert. Dazu wird zunächst die Entscheidung für ein dokumentbasiertes Datenbanksystem begründet. Danach wird das verwendete Datenbanksystem *MongoDB* vorgestellt. Darauf folgend wird im Abschnitt 4.1.3 darauf eingegangen, wie die ermittelten Kategorien in *MongoDB-Collections* aufgeteilt werden.

4.1.1 Entscheidung dokumentbasiertes Datenbanksystem

Wie in der Projektvorstellung im Unterkapitel 2.1 auf Seite 5 erläutert, besteht das Projekt momentan aus zwei Teilen. Die Schnittstelle dieser Teile ist die gemeinsame Datenbank. Daher müssen bei der Wahl des Datenbanksystems die Anforderungen beider Teilprojekte betrachtet werden.
Aus dem Grund, dass im zweiten Teil mit SCPI-Befehlen gearbeitet wird, die unterschiedliche Datentypen haben können, muss es eine Möglichkeit geben, dies flexibel zu lassen.
In dieser Arbeit, dem Teil der *Datenmanagementsoftware*, müssen große Datenmengen in der Datenbank abgespeichert werden. Deshalb ist es das Ziel, keinen unnötigen Speicherplatz durch ein unpraktisches Datenbankkonzept zu benötigen. Die Möglichkeit soll offen stehen neue Eigenschaften hinzufügen zu können. Aus diesem Grund

musste bei der Entscheidung ebenfalls betrachtet werden, ob dies in dem Datenbanksystem leicht ermöglicht wird. sollen diese leicht in die Datenbank einbinden zu können.
Da es viele Abhängigkeiten zwischen den Daten gibt, musste dies für eine Entscheidung ebenfalls betrachtet werden.
Für das Gesamtprojekt wurden dokumentenorientierte, graphenorientierte und SQL-Datenbanken miteinander verglichen. Dazu wurde speziell auf die Anforderungen an das Datenbanksystem geachtet. Gemeinsam wurde sich dann für *MongoDB*, eine dokumentenorientierte Datenbank, entschieden, da diese die Anforderungen an die Datenbank am besten abdecken kann.

4.1.2 *MongoDB*

MongoDB ist eine sogenannte NoSQL-Datenbank. Das bedeutet, dass sie keiner relationalen Datenbank entspricht. Stattdessen ist *MongoDB* eine dokumentbasierte Datenbank.
Um die verwendeten Komponenten einer dokumentbasierten Datenbank zu verdeutlichen, werden diese im Vergleich zu jenen eines relationalen Datenbankmanagementsystems erläutert. Wie in Abbildung 4.1 zu sehen ist, ist eine *Collection* ein Äquivalent zu einer Tabelle eines relationalen Datenbankmanagementsystems und eine Zeile einer relationalen Datenbank ist ähnlich zu einem Dokument von *MongoDB*.

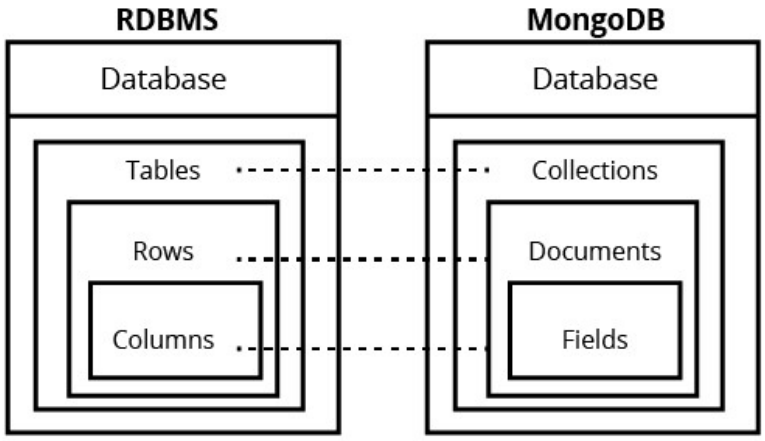

Abbildung 4.1: Komponenten vom RDBMS zu *MongoDB* [4]

Auch wenn man den verschiedenen Komponenten Äquivalente zuordnen kann, unterscheiden sie sich in ihrer Funktion. Im Folgenden wird auf wichtige Aspekte von *MongoDB* eingegangen.

Datenbank

Die Datenbank ist ein physischer Speicher für die Abspeicherung von *Collections*. Dabei hat jede Datenbank einen eigenen Satz an Dateien im Dateisystem. Normalerweise verwaltet ein einzelner *MongoDB*-Server mehrere Datenbanken. [5]

Dokument und *Collection*

Ein Dokument besteht aus einem Datensatz an *key-value*-Paaren und hat ein dynamisches Schema. Ein dynamisches Schema bedeutet, dass Dokumente in der selben *Collection* nicht den gleichen Datensatz haben müssen. Eine *Collection* ist eine Ansammlung von *MongoDB*-Dokumenten. Stattdessen können die Dokumente unterschiedliche *key-value*-Paare besitzen. Obwohl *MongoDB* schemalos ist, können durch die *Schema Validation* Anforderungen an die Dokumente gesetzt werden. Auf die *Schema Validation* wird im weiteren Verlauf näher eingegangen. [5]

BSON

BSON steht für „Binary JSON“. In *MongoDB* werden die Daten in den Dokumenten intern in *BSON* abgespeichert. *BSON* ist dabei eine Erweiterung zu *JSON*. Binäre Daten in eine *JSON* zu übertragen ist mit viel Aufwand verbunden. Außerdem unterstützt *JSON* weniger Datentypen als *BSON*. In *JSON* gibt es zum Beispiel keinen Datentyp für einen Datumswert. Ein Datum kann zwar als Timestamp übertragen werden, jedoch geht dabei die Information der Zeitzone verloren. In *BSON* ist dies wiederum möglich. [6]

Embedded Documents

Ein *Embedded Document* ist ein Teildokument, welches in ein Top-Level-Dokument mit eingebunden wird. Bei einer relationalen Datenbank, bei der die Normalformen eingehalten werden, werden viele Informationen in extra Tabellen ausgelagert, wodurch *Joins* notwendig werden. Dies ist durch ein *Embedded Document* nicht notwendig.[6]
Daher kann für verschachtelte Daten verwendet werden, die nicht mehrfach abgespeichert werden sollen. Ein Beispiel hierfür wären die Daten eines Ansprechpartners einer Firma, wenn dabei die Eigenschaften der Firma eine *Collection* bilden.

Schema Validation

MongoDB ist schemalos und die Daten brauchen nicht zwingend einem bestimmten Format entsprechen. Daher ist es nicht zwingend notwendig eine Schemavalidation zu erstellen. Sollen Daten beim Aktualisieren oder Einfügen in die Datenbank dennoch auf Vollständigkeit und das Einhalten des richtigen Datentyps überprüft werden, so werden den *Collections* sogenannte Schemavalidatoren hinzugefügt.

Damit in der *Datenmanagementsoftware* sichergestellt werden kann, dass die Daten vollständig und im richtigen Datenformat abgespeichert werden, werden Schemavalidatoren benötigt. Damit kann auch sichergestellt werden, dass beim Auslesen der Daten diese in dem notwendigen Format sind und diese dementsprechend auf der Benutzeroberfläche angezeigt werden können.
Die Schemavalidatoren können als Parameter beim Erstellen einer neuen *Collection* in der Funktion „db.create_collection“ mit angeben werden.
Um diese Art der Überprüfungsregeln einer bestehenden *Collection* hinzuzufügen wird die Methode „collMod“ verwendet. [7]

Um die Syntax der Schemavalidation zu erklären, wurde im Listing 4.1 die Python-Methode zur Generierung einer *Collection* mit einer Schemavalidierung dargestellt. *MongoDB* unterstützt die JSON-Schemavalidierung, diese wird mit dem Operator „$jsonSchema“ in Zeile 4 eingebunden.
Datentypen werden über das Keyword „bsonType“ angegeben.
Ebenfalls können in einer Liste alle Eigenschaften aufgelistet werden, die notwendig für die *Collection* sind. Dies wird mit dem Keyword „required“ in Zeile 6 definiert.
Alle Eigenschaften der *Collection* werden unter dem Keyword „properties“ aufgelistet.

```
1  {
2  db.create_collection(<Collection-Name>, validator=
3      {
4          "$jsonSchema": {
5              "bsonType": "object",
6              "required": [<Eigenschaften die Notwendig sind>],
7              "properties": {
8                  <Eigenschaft>: {
9                      "bsonType":    <Datentyp>,
10                     "description": <Beschreibung>
11                 },
12             }
13 }
```

Listing 4.1: Syntax einer Schemavalidation

4.1.3 Konzeptionierung der *Collection*

Die in Unterkapitel 3.1 ab Seite 11 aufgelisteten Kategorien können in *Collections* aufgeteilt werden. In der Abbildung 4.2 werden dabei in den Rechtecken die *Collections* dargestellt. Da manche Kategorien Eigenschaften mit Informationen aus anderen Kategorien besitzen, werden die *Collections* miteinander verknüpft.
In einem Messsetup werden mehrere Geräte gespeichert. Daher besitzt die *Collection* für das Messsetup eine Liste an IDs der Geräte *Collection*. Die *Collections* Verleih, Verwertung und Reparatur, sowie Kalibrierung und Wartung wurden durch einen großen Kasten zusammengefasst, da diese die gleichen Abhängigkeiten besitzen.

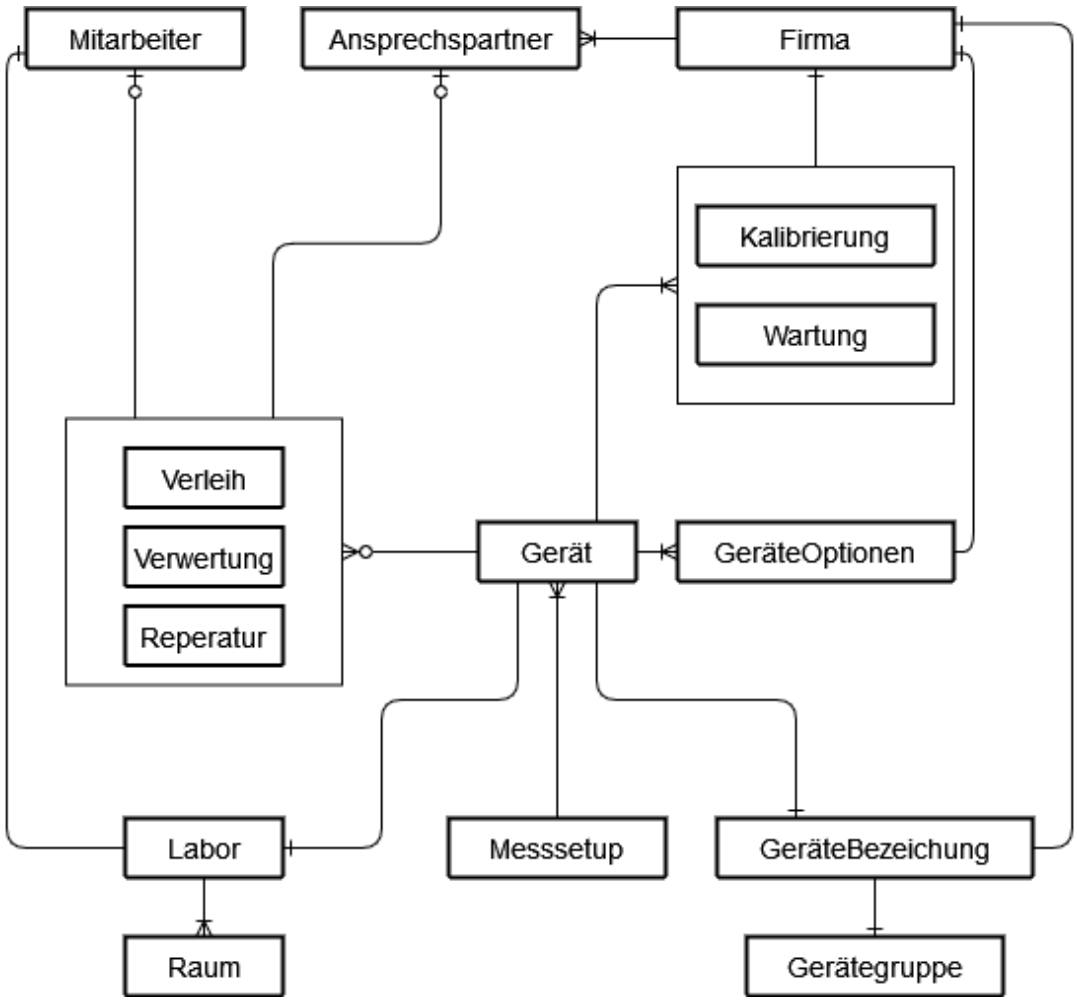

Abbildung 4.2: Übersicht der *Collections*

4.2 Konzept der Benutzeroberfläche

In diesem Unterkapitel werden die wichtigsten Benutzeroberflächen konzipiert. Bei der Konzeptionierung wurde die Anforderung der Bilingualität berücksichtigt. Zur Visualisierung sind alle in diesem Unterkapitel gezeigten Grafiken nur für den deutschen Sprachraum erstellt worden.

4.2.1 Entwurf der Benutzeroberflächen

Zur Visualisierung der Entwürfe der jeweiligen Benutzeroberflächen wurden diese mit Hilfe des *Qt-Creator*s erstellt. Durch die Möglichkeit Elemente mittels „Drag and Drop“ einzufügen ergibt sich eine unkomplizierte und schnelle Erstellung eines Schemas der Oberfläche. Ein weiterer Grund war, dass es weiterhin die Möglichkeit bietet, diese Oberflächen weiter zu verwenden, um das Projekt in Form einer Desktopanwendung umzusetzen. Hierfür können die Dateien mit wenig Aufwand leicht in PyQt übersetzt werden, wodurch die Desktopanwendung mit Python umgesetzt werden könnte.
Im Rahmen dieser Bachelorarbeit konnte kein Tool gefunden werden, in dem einerseits das Schema der Oberfläche für eine Webanwendung leicht designt werden kann und andererseits dies auch für die Umsetzung als Sourcecode genutzt werden kann. Auf Grund dessen wurde der *Qt-Creator* als Tool ausgewählt.

Es wurden dabei verschiedene Benutzeroberflächen entworfen, die weiter thematisiert werden. Dazu wird im Folgenden zuerst auf die Oberfläche zur Anmeldung eingegangen und danach die Visualisierung der grundlegenden Informationen eines Geräts erläutert.
Da der Fokus auf der Kalibrierung liegt, werden anschließend die Benutzeroberflächen für die verschiedenen Schritte der Kalibrierung vorgestellt.
Daraufhin wird die Oberfläche zur Erzeugung von Messsetups beschrieben.

Oberfläche zur Anmeldung

Wie in Abschnitt 2.2 auf Seite 6 erläutert, haben verschiedene Benutzergruppen unterschiedliche Rechte. Dies soll durch eine Anmeldung gesteuert werden. Da es für jeden Nutzer möglich sein soll, einen lesenden Zugriff auf die hinterlegten Informationen zu haben, ist hierfür eine Anmeldung nicht zwingend notwendig. Daher wird die Möglichkeit eingeplant, über einen Button ohne eine Anmeldung fortzufahren. Die eigentliche Benutzeranmeldung soll über die jeweiligen LDAP-Daten möglich sein.

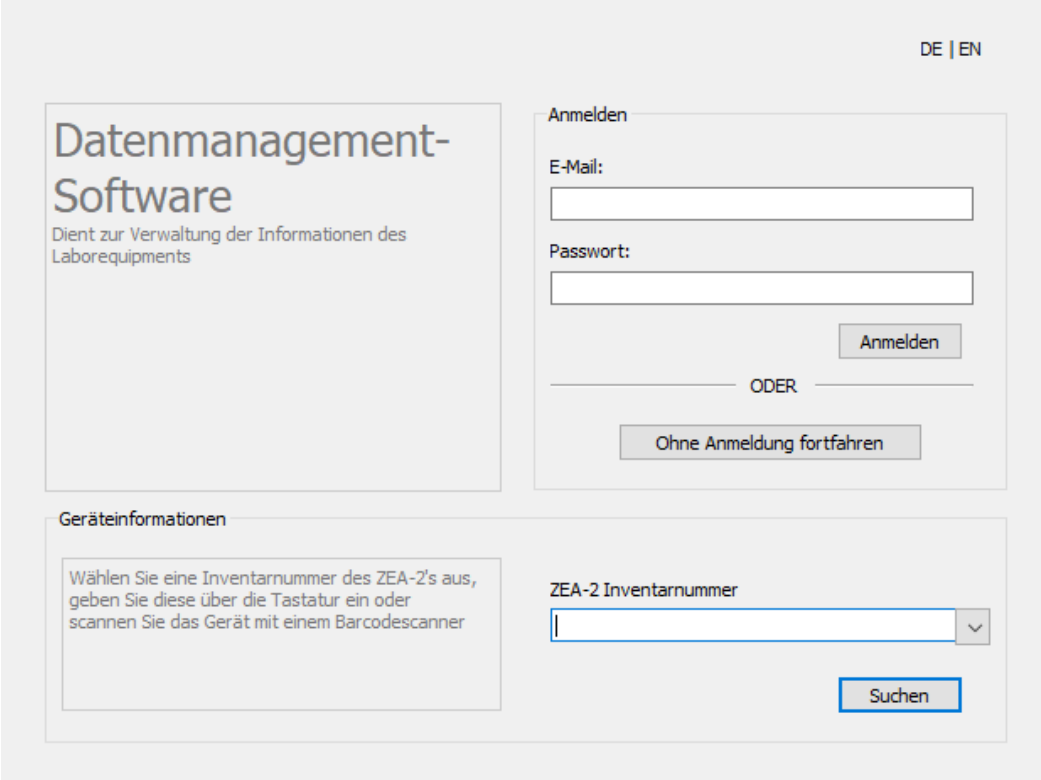

Abbildung 4.3: Benutzeroberfläche für die Anmeldung

In der Abbildung 4.3 ist der Entwurf der Oberfläche zu sehen. Hierbei sind die zuvor erläuterten Eigenschaften im rechten oberen Bereich der Abbildung zu finden. Der linke obere Bereich wird für eine allgemeine Beschreibung der Software vorgehalten. Eine weitere Funktionalität der Oberfläche zur Anmeldung ist es, direkte Informationen zu einem Gerät zu erhalten, indem die Inventarnummer des *ZEA-2*'s angegeben wird. Dies ist in der Abbildung 4.3 unten dargestellt. Die Inventarnummer kann dabei über die Tastatur eingegeben werden, über die Checkbox gesucht werden oder indem das Gerät mit einem Barcodescanner eingescannt wird. Diese Funktionalität wird durch den unteren Bereich der Abbildung 4.3 realisiert.
Im folgenden Abschnitt wird auf die gerade genannte Übersicht der Geräteinformationen näher eingegangen.

Übersicht der Informationen eines Gerätes

Die Informationen zu einem spezifischen Gerät, die in Abschnitt 3.1.1 erwähnt wurden, kann sich der Benutzer über zwei Wege anzeigen lassen. Entweder kann der Benutzer die Inventarnummer in der Oberfläche zur Anmeldung, das im vorherigen Abschnitt gezeigt wurde, direkt eingeben oder durch eine spezifische Auswahl im Menüpunkt „Geräte“.
Dabei werden auf der Oberfläche alle relevanten Informationen zu dem jeweiligen Gerät dargestellt. Zudem wird ein Bild des Geräts abgebildet und es soll möglich sein, die Kalibrierungsberichte über das Klicken eines Buttons anzeigen zu lassen. In Abbildung 4.4 ist die Oberfläche zu sehen. Dabei ist die Maske der Oberfläche zur Anzeige der gesammelten Informationen für alle Geräte identisch und die grau hinterlegten Bereiche enthalten die Geräteinformationen, die vom System abgerufen werden müssen. Eine sehr wichtige Information des Geräts ist, ob das Gerät kalibriert ist. Daher wird dies, wie in der Abbildung zu sehen ist, durch einen Balken über die komplette Breite des Fensters verdeutlicht.

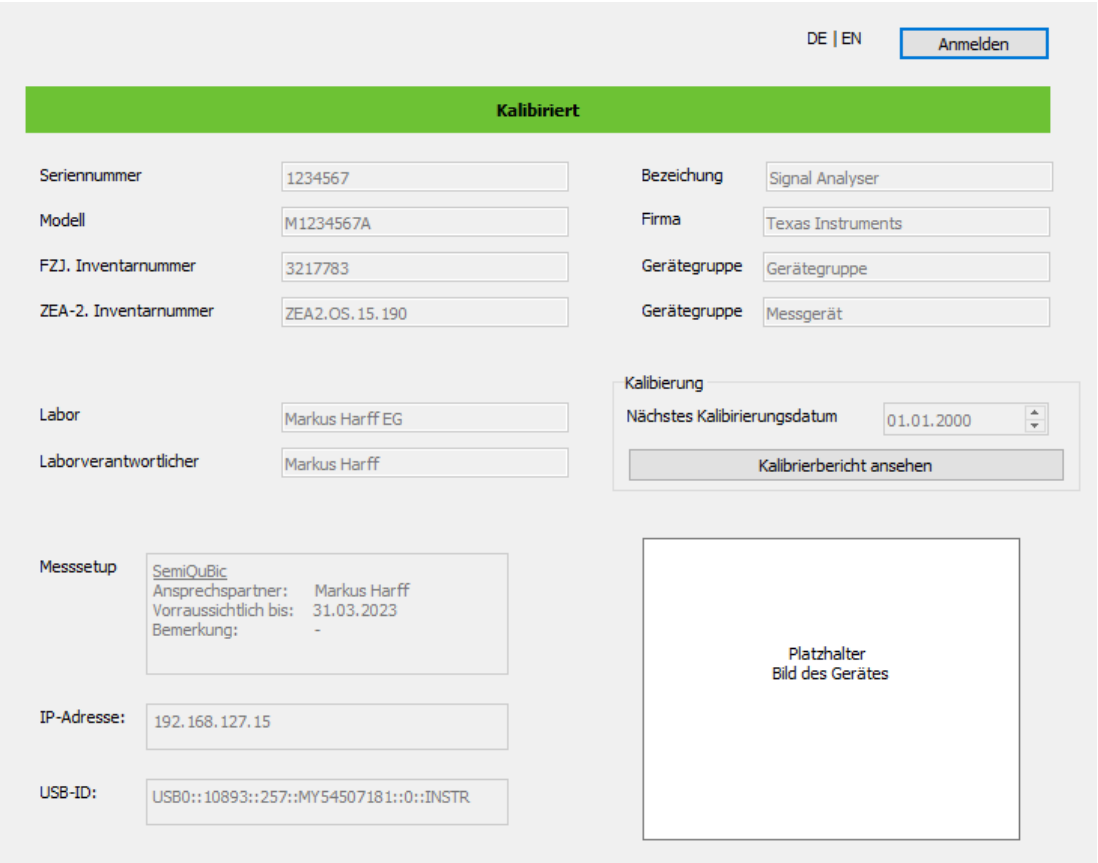

Abbildung 4.4: Benutzeroberfläche für die gerätespezifischen Informationen

Dabei kann die Anzeige verschiedene Bedeutungen mit entsprechenden unterschiedlichen Farben aufweisen. Wenn das Gerät kalibriert ist, wie hier veranschaulicht, wird diese Anzeige grün leuchten und den Text „kalibriert“ anzeigen. Für ein Gerät, das sich im Kalibrierungsprozess befindet, ist die Anzeige orange. Für den Fall, dass das Gerät nicht rechtzeitig kalibriert wurde und das geplante nächste Kalibrierungsdatum schon überschritten wurde, wird die Anzeige rot hinterlegt mit einer Warnmeldung. Der letzte zu betrachtende Fall ist, dass das Gerät aussortiert wurde. In diesem Fall hinterlegt die Anzeige dann grau mit einer entsprechenden Meldung.

Kopfzeile mit einer Sidebar

Die Oberflächen besitzen alle ein Kopfzeile. In dieser wird, wie in der Abbildung 4.5 zu sehen ist, mittels des Namens angezeigt, welcher Benutzer gerade eingeloggt ist. In Form eines Togglebuttons kann die Sprache von Deutsch auf Englisch verändert werden. Dies ist in der Abbildung durch die Angabe „DE | EN“ repräsentiert.

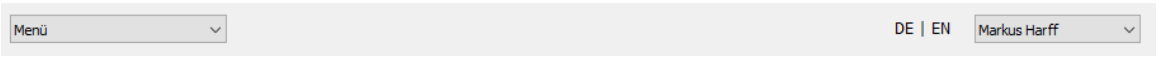

Abbildung 4.5: Kopfzeile der Oberflächen

Der Button links mit dem Text „Menü“ stellt eine ausklappbare Sidebar da. Der Vorteil davon, dass sie aus- und einklappbar ist, ist dass die Darstellung der restlichen Seite mehr Platz bekommt. Außerdem wird laut Anwender nicht so häufig das Menü gewechselt, dass die Sidebar links fest stehen sollte. In Abbildung 4.6 ist die Sidebar ausgeklappt dargestellt. Dabei wurde zu den wichtigsten Bereichen ein Menüpunkt erstellt. Wenn der Benutzer auf einen dieser Menüpunkte drückt, soll der ausgewählte Bereich angezeigt werden und die Sidebar sich einklappen.

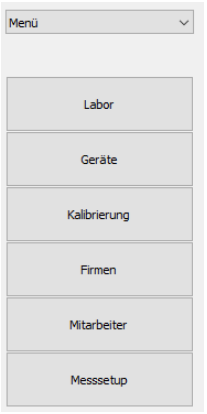

Abbildung 4.6: Sidebar der Oberfläche

Kalibrierung der Geräte

Um überhaupt einen Einfluss auf den zuletzt angesprochenen Balken des Kalibrierungsstatus nehmen zu können, muss eine Ansicht generiert werden, in der der Benutzer ein neuen Kalibrierungsantrag erstellen kann und eine Ansicht, in der der Benutzer Informationen zu bestehenden Kalibrierungsanträgen bekommt. Diese beiden Ansichten wurden mittels eines *Tab Widgets* in der Oberfläche umgesetzt.
Der erste Tab „Kalibrierungsübersicht“ listet alle erstellen Kalibrierungsanträge auf. Dort wird der Benutzer dann die Möglichkeit haben, durch ein Doppelklick genauere Informationen zu diesem Antrag angezeigt zu bekommen. Wie das Schema dieses Tabs ausschaut, wird im Abschnitt „Übersicht der Kalibrierungen“ auf Seite 27 thematisiert, in dem darauf folgenden Abschnitt „Detailansicht einer Kalibrierung“, wird dann näher auf die Darstellung eingegangen der detaillierten Ansicht des Kalibrierungsauftrags.
Der Tab „Neuer Kalibrierungsantrag“ dient dazu, durch eine Auswahl von Geräten einen neuen Kalibrierungsantrag zu stellen. Darauf wird im folgenden Abschnitt detaillierter eingegangen.

Neuer Kalibrierungsauftrag

Damit ein Kalibrierungsantrag gestellt werden kann, müssen vorab die Geräte ausgewählt werden, die zu kalibrieren sind. Wie in Unterkapitel 3.2 auf Seite 14 analysiert, sollte es möglich sein, die zu kalibrierenden Geräte aus einer Liste von Geräten auszuwählen.

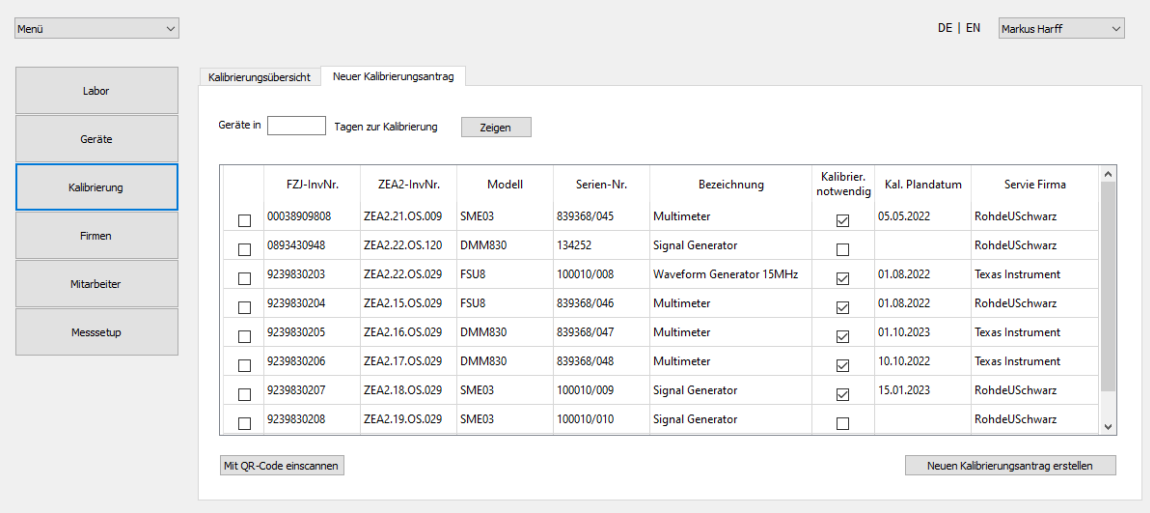

Abbildung 4.7: Benutzeroberfläche für die Erstellung eines Kalibrierungsantrages

Dabei kann der Benutzer eine Anzahl an Tagen eingeben, damit nur die Geräte, die bis dahin kalibriert werden müssen, angezeigt werden. Außerdem kann es sein, dass Geräte, die normalerweise nicht kalibriert werden, ausnahmsweise ebenfalls kalibriert werden sollen. In diesem Fall werden nicht nur die Geräte angezeigt, die bis dahin kalibriert werden müssen, sondern auch alle Geräte, die normalerweise nicht kalibriert werden.
Des Weiteren soll es möglich sein, die Geräte, die ausgewählt werden sollen, auch mittels eines QR-Code-Scanners einzuscannen. Dazu dient der Button „Mit QR-Code einscannen“. Der Benutzer drückt diesen und kann danach die Geräte einscannen. Diese werden dann in der Tabelle als ausgewählt markiert.
Nach einer Auswahl der zu kalibrierenden Geräte müssen diese noch durch den Button „Neuen Kalibrierungsantrag erstellen“ bestätigt werden. Mit dem Drücken soll dann eine Datei erstellt werden, die alle zu kalibrierenden Geräte auflistet. Diese Datei soll dann verwendet werden, um bei den Firmen den Auftrag zum Kalibrieren stellen zu können.

Übersicht der Kalibrierungen

Nach dem eine Kalibrierung gestartet wurde, indem der Benutzer einen neuen Kalibrierungsantrag erstellt hat, werden diese Informationen abgespeichert und es kommt zu einen neuen Eintrag in der Übersicht der Kalibrierungen.

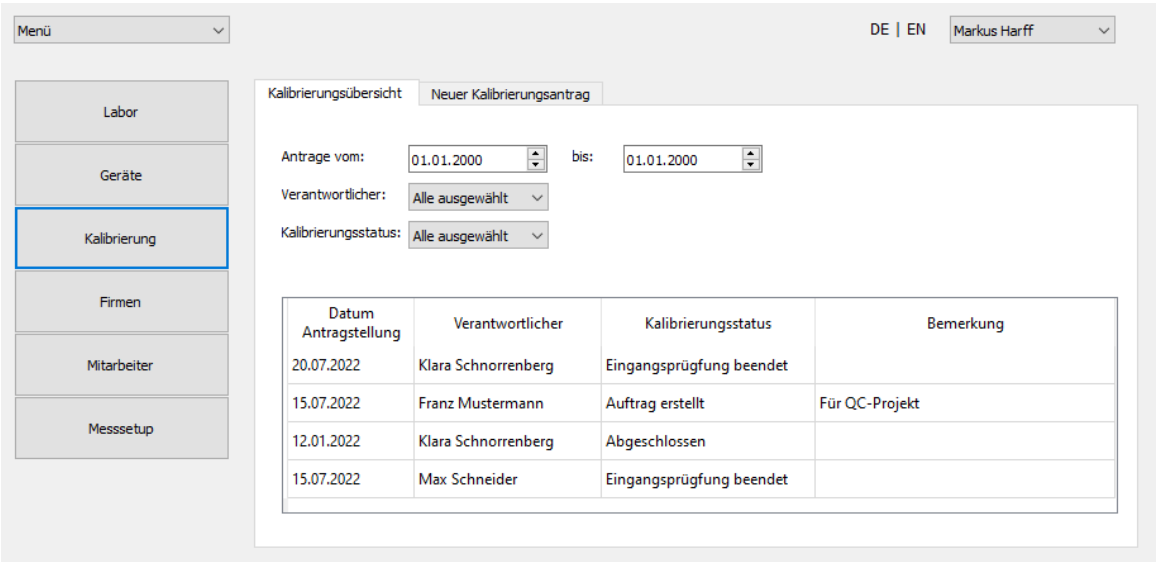

Abbildung 4.8: Benutzeroberfläche für eine Übersicht an Kalibrierungsaufträgen

In dieser Ansicht werden alle Kalibrierungen sortiert nach dem Datum der Antragstellung aufgelistet. Neben dem Datum werden die Informationen über den Verant-

wortlichen der Kalibrierung, den Status und einer möglichen Bemerkung angegeben. Der Status der Kalibrierung hat dabei drei verschiedene Zustände:

1. Auftrag erstellt
   Neuer Kalibrierungsantrag wurde erstellt

2. Eingangsprüfung beendet
   Geräte wurden geprüft und die Ergebnisse wurden eingetragen

3. Abgeschlossen
   Alle Geräte wurden kalibriert, repariert oder aussortiert und entsprechende Dokumente hochgeladen

Wenn auf eine Zeile doppelt geklickt wird, öffnet sich eine spezifischere Anzeige der Kalibrierung.

Detailansicht einer Kalibrierung

In der Detailansicht einer Kalibrierung werden alle Informationen zu der ausgewählten Kalibrierung angezeigt. Dazu gehört das Datum, der Antragsteller und alle Geräte die kalibriert werden sollen bzw. wurden. Über diese Oberfläche hat der Benutzer ebenfalls die Möglichkeit, die Daten anpassen.
Wie im vorherigen Abschnitt angeschnitten, gibt es drei Zustände der Kalibrierung. Falls die spezifischere Anzeige durch einen doppelklick geöffnet wird, wird dann je nach Zustand eine andere Oberfläche angezeigt.
Wenn der Zustand auf „Auftrag erstellt“ steht, bedeutet dies, dass der Benutzer die Geräte für die Kalibrierung ausgewählt hat. Sobald für jedes Gerät dann eingetragen wurde, ob die Eingangsprüfung erfolgreich war oder nicht, wird der Status auf „Eingangsprüfung beendet“ gesetzt. Solange die Eingangsprüfung noch nicht vollständig beendet wurde, wird dem Benutzer eine Oberfläche angezeigt, in der über Checkboxen eine Auswahl getroffen wird, ob die Eingangsprüfung bestanden wurde oder nicht.

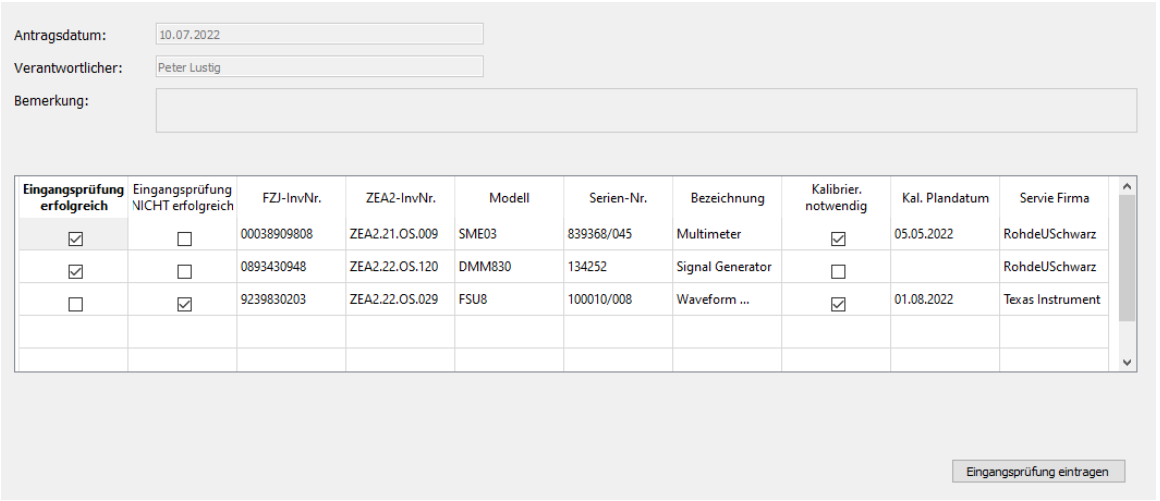

Abbildung 4.9: Benutzeroberfläche des Kalibrierungsantrages für die Eingangsprüfung-Eintragung

Sobald die Eingangsprüfung vollkommen abgeschlossen ist, wird es in einer anderen Oberfläche die Möglichkeit geben, entsprechende Dokumente hochzuladen.

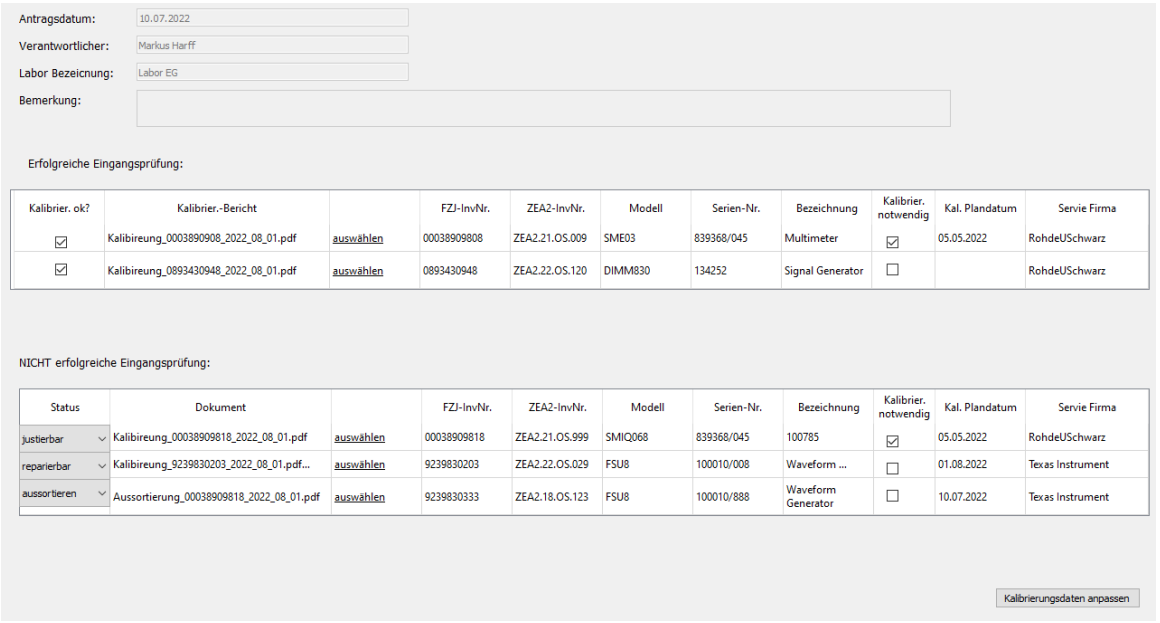

Abbildung 4.10: Ausschnitt aus der Benutzeroberfläche für die Einbuchung der Kalibrierung

Messsetup

Die Oberfläche des Messsetups ist genauso wie die Oberfläche der Kalibrierung (Abschnitt „Kalibrierung der Geräte“ auf S.26) in zwei zwei Tabs aufgeteilt.
Der eine Tab gibt eine Übersicht der Messsetups und in dem anderen Tab kann ein neues Messsetup erstellt werden.

Neues Messsetup erstellen

Eine weitere Funktionalität ist das Erstellen eines Messsetups. Hierfür wird ein Name für das Messsetup festgelegt, wie in dem Abschnitt zu der Funktionalität des Messsetups 2.3.2 auf Seite 8 beschrieben. Zusätzlich muss, wie in der Abbildung 4.11 zu erkennen, ein Verantwortlicher angegeben werden. Dies kann ein beliebiger Mitarbeiter sein.

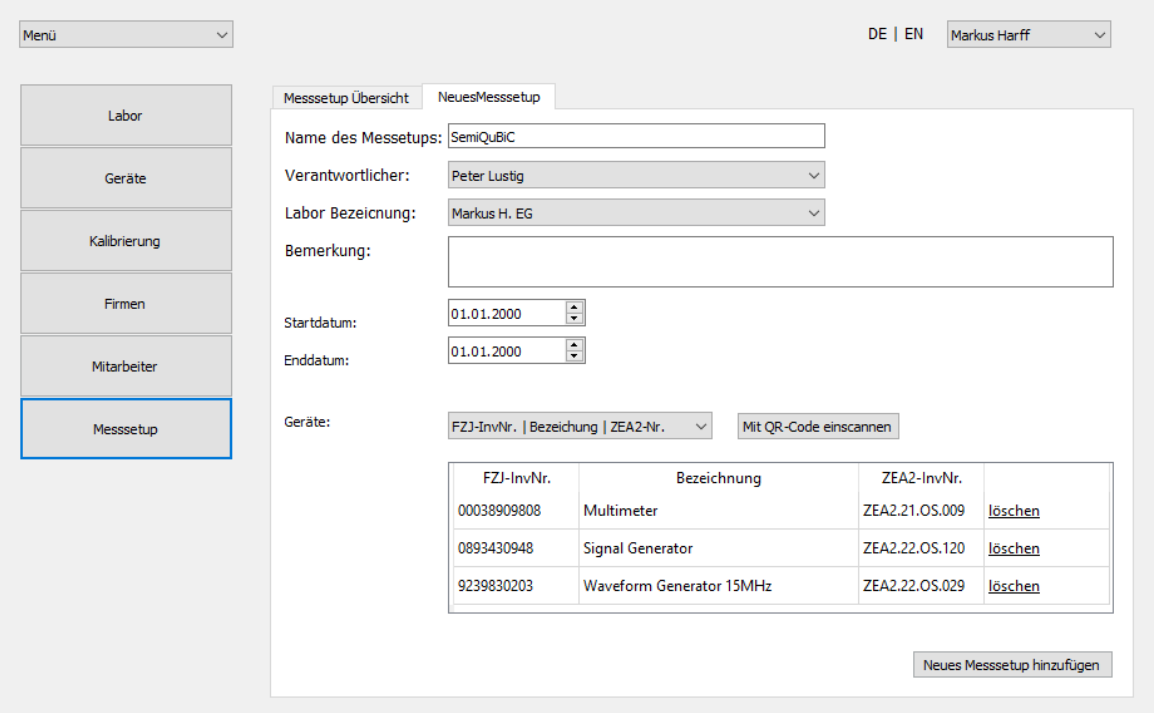

Abbildung 4.11: Benutzeroberfläche für das Erstellen eines neuen Messsetups

Zudem wird ein Labor ausgewählt, dem das Messsetup zugeordnet werden soll. Der Benutzer gibt sowohl einen Start- als auch Endtermin für das Messsetup an. Optional kann zudem auch noch eine Bemerkung hinzugefügt werden.
Die Geräte können danach mit Hilfe einer Combo Box hinzugefügt werden.

In dieser können die Geräte ausgewählt werden. Oder es besteht die Möglichkeit, dass die Geräte mit einem QR-Code-Scanner eingescannt werden. Dafür wird auf den Button „Mit QR-Code einscannen“ gedrückt. Danach können dann die Geräte eingescannt werden. Alle Geräte, die entweder über die Combo Box oder dem QR-Code-Scanner ausgewählt wurden, werden in einer Tabelle aufgelistet. In dieser besteht dann die Möglichkeit, dieses Gerät durch Drücken auf den „löschen“-Button wieder zu entfernen. Sobald alle Daten eingegeben wurden, wird dies mit dem Button „Neues Messsetup“ bestätigt.

Übersicht der Messsetups

Die Übersicht der Messetups wird tabellarisch angegeben, indem der Messsetupname, der Verantwortliche und das Start- und Enddatum aufgelistet werden. Der Benutzer hat dabei durch eine Auswahl an Checkboxen die Möglichkeit, zu filtern, welche Messsetups angezeigt werden sollen. Dabei wird unterschieden in Messsetups, die schon beendet sind, oder Messsetups, die momentan bestehen, und Messsetups, die zukünftig eingesetzt werden sollen.

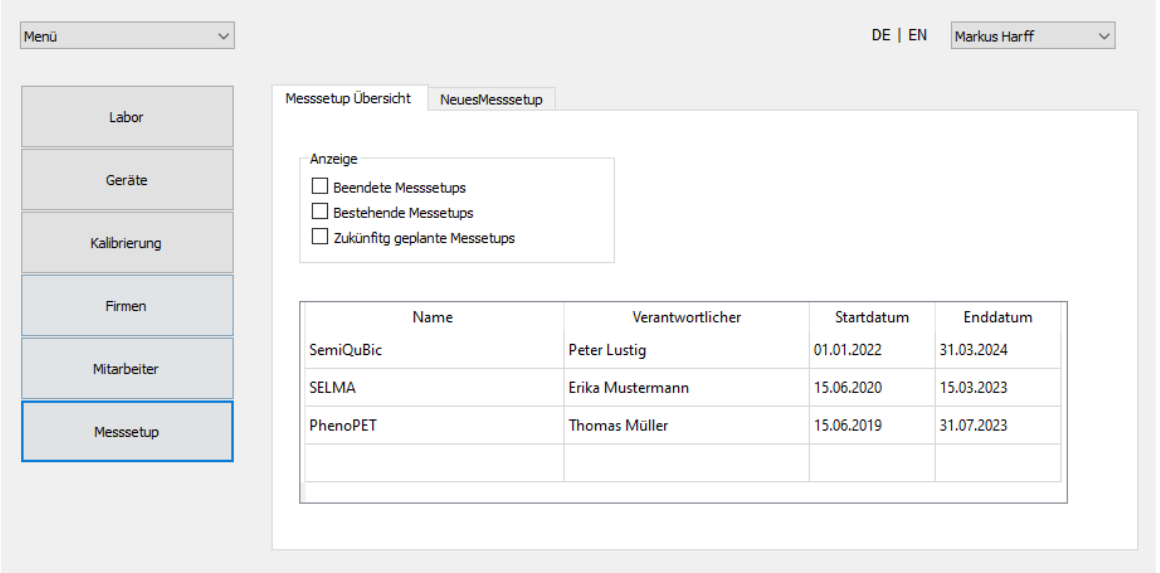

Abbildung 4.12: Benutzeroberfläche für eine Übersicht der Messsetups

Durch ein Doppelklick auf ein Messsetup wird eine detaillierte Ansicht des Messsetups angezeigt, auf welche nun im Folgenden weiter eingegangen wird.

Detailansicht eines Messsetups

Die Oberfläche für eine detaillierte Ansicht des Messetups ist ähnlich zu dem Hinzufügen eines neuen Messetups. Wie das Schema dieser Oberfläche aussieht, wird in der Abbildung 4.11 auf Seite 30 dargestellt. Die Oberfläche der Detailansicht hat zusätzlich die Möglichkeit Messprotokolle hochzuladen bzw. zu öffnen.

4.2.2 Unterschied zwischen Web- und Desktopanwendung

In der Aufgabenanforderung wurde offen gehalten, ob das Projekt als Web- oder Desktopanwendung umgesetzt werden soll. Deshalb werden diese beiden Möglichkeiten im Folgenden anhand von verschiedenen Faktoren betrachtet und gegeneinander abgewogen. Festgelegt sind für diese Betrachtungen, dass MongoDB als Datenbank genutzt wird und der Datenbankserver auf einem Server des *ZEA-2*'s laufen soll.

Erreichbarkeit und Update der Software

Da sowohl Webserver als die Datenbank auf einem internen Server des *ZEA-2*s gehostet würden, sind diese nur über das Intranet des Forschungszentrums zu erreichen. Der Grund dafür ist, dass sich der Server hinter einer Firewall befinden würde und nicht extern freigegeben wird. Von außerhalb wäre dann eine VPN-Verbindung notwendig.
Die Realisierung mittels einer Webanwendung bietet allerdings den Vorteil, dass auf Grund des selben Systems die Wahrscheinlichkeit für Verbindungsabbrüche vermutlich minimal wäre. Ein weiterer Vorteil ist die direkte Erreichbarkeit, da keine Installation auf dem jeweiligen Desktoprechner nötig ist und so ein Zugriff von verschiedenen Geräten jederzeit direkt möglich ist. Bei einer Desktopanwendung muss der Benutzer die Anwendung zu Beginn installieren und möglicherweise auch die notwendigen Systemanforderungen erfüllen. Zusätzlich muss sichergestellt werden, dass neue Softwareversionen auf die jeweiligen Rechner überspielt werden, um neue Features nutzen zu können. Eventuell sind Interaktionen nötig, da nicht jeder Nutzer die Administrationsrechte für den Rechner besitzt. Somit könnten Updates zumindest einen gewissen Teil der internen IT-Administration benötigen.
Bei einer Webanwendung erfolgen die Änderungen automatisch ohne jegliche Interaktion des Benutzers, sobald die Seite aktualisiert wird. Allerdings muss bei der Webanwendung eine dauerhafte Internetverbindung bestehen. Sobald dies nicht der Fall ist, wird die Verbindung zur Webanwendung verloren.

In diesem gleichen Szenario bietet die Desktopanwendung den Vorteil, dass die bereits vom der Datenbank abgerufenen Daten weiterhin visualisiert werden können. Eine Anpassung der Daten ist in beiden Fällen nicht möglich.

Update der Software

Auf Grund der lokalen Kopie bei einer Desktopanwendung kann es vorkommen, dass ein Update der Software durchgeführt werden muss. Hierbei sollte der Nutzer aktiv informiert werden und.
Sofern ein Update der Webanwendung durchgeführt wurde, wird dem Benutzer beim nächsten Öffnen die neuste Version angezeigt.

Umsetzbarkeit

Des Weiteren muss der Faktor der Umsetzbarkeit betrachtet werden. Hierbei wird sich im folgenden hauptsächlich auf die Entwicklungsdauer bzw. den Mehraufwand der einzelnen Optionen fokussiert.
Da es keine festen Vorgaben gibt, welche Geräte für die Nutzung der Software vorgesehen werden, werden je nach Anwendungsfall die Möglichkeiten betrachtet.
Bei einer Webanwendung spielt der Aufbau der Oberflächen eine wichtige Rolle, da dort speziell darauf geachtet werden muss, wie diese auf verschiedenen Geräten angezeigt werden. Dabei hat die Desktopanwendung den Vorteil, dass dort nur die Skalierung betrachtet werden muss, bzw. eine minimale Auflösung als gegeben angesehen werden kann.
Bei der Desktopanwendung muss jedoch beachtet werden, für welches Betriebssystem die Anwendung sein muss. Sobald es für mehrere Betriebssysteme ermöglicht werden muss, muss dies in der Umsetzung speziell beachtet werden.
Da die Benutzeroberflächen mittels des Qt-Creators entworfen wurden, und dann die Umsetzung in Qt oder mit PyQt durch Python naheliegend wäre, ist in diesem Fall jedoch die Betrachtung der unterschiedlichen Betriebssysteme größtenteils vernachlässigbar. Innerhalb des ZEA-2 werden derzeit nur die Betriebssysteme Linux und Windows verwendet. Die nötigen Änderungen zwischen diesen beiden würden sich dabei auf einen geringen Bereich beschränken. Bei der Webanwendung muss dafür speziell betrachtet werden wie das Frontend und Backend aufgebaut wird. Dabei muss speziell überprüft werden, ob in dem Projekt Funktionalitäten benötigt werden, welche von den im *ZEA-2* genutzten Browsern nicht unterstützt werden. In diesem Fall müssten Unterschiede in der Realisierung für diese Browser umgesetzt werden.

Benutzerfreundlichkeit

Die Benutzerfreundlichkeit ist ein wichtiger Aspekt zur Differenzierung. Dazu zählen, ob die Anwendung leicht bedienbar ist und wie benutzerfreundlich der Start bzw. die Installation der Anwendung ist. Zu diesem Aspekt zählt ebenfalls der vorher angesprochene Faktor der Erreichbarkeit.
Wenn dabei die Webanwendung und die Desktopanwendung verglichen werden, hat die Webanwendung zur Desktopanwendung zum einen den Vorteil, dass der Benutzer sich um keine Installationen vorab kümmern muss, als auch dass Updates nicht dem Benutzer überlassen werden. Die Nutzung der Anwendung unterscheidet sich kaum, da Nutzer sowohl das Öffnen von Programmen als auch das Öffnen eines Links mittels Browsers gewöhnt sind. Ein Vorteil der Desktopanwendung hingegen könnte sein, dass es möglich ist, die Software in den Autostart zu integrieren. Dadurch wäre die Umgebung am Rechner beim Starten des Rechners bereits gegeben. Ein ähnliches Verhalten kann bei der Webanwendung erzielt werden, wenn die entsprechende Seite als Startseite eingerichtet wird.

4.2.3 Entscheidung

Um die zuvor beschriebenen Bereiche vergleichen zu können, wurden der Vergleichsfaktor kategorisiert. Hierbei bedeutet grün, dass ein betrachteter Aspekt gut erfüllt wurde. Analog dazu steht rot für eine schlechtere Erfüllung der Kriterien. In der Tabelle 4.1 wird schematisch die Desktopanwendung und Webanwendung farblich dargestellt und dabei die oben verglichenen Faktoren gegenübergestellt.

Tabelle 4.1: Vergleich von Web- und Desktopanwendung

	Desktopanwendung	Webanwendung
Erreichbarkeit		
Update der Software		
Umsetzbarkeit		
Benutzerfreundlichkeit		

Wie in dem farblichen Vergleich zu sehen ist sind auf der Seite für die Webanwendung wesentlich mehr Faktoren grün gefärbt. Die Ausnahme liegt lediglich im Bereich der Umsetzbarkeit, wo die Webanwendung wesentlich schlechter bewertet wurde als die Desktopanwendung. Da alle anderen Faktoren bei der Webanwendung besser abschneiden und die Umsetzbarkeit nur im zeitlichen Sinne negative Auswirkungen hat, ist es sinnvoll, mehr Zeit in die Realisierung zu investieren, dann aber langfristig von den positiven Faktoren einer Webanwendung zu profitieren.

4.3 Webanwendung-Komponenten

Wie im Abschnitt 4.2.3 erläutert, wird die Software in Form einer Webanwendung umgesetzt. Um eine Webanwendung zu implementieren, werden verschiedene Komponenten benötigt. Auf diese wird im Folgenden eingegangen. Zuerst wird ReactJs mit Bootstrap erklärt, da dieses für das Frontend benutzt wird.
Anschließend folgt ein Vergleich des MERN- und FARM-Stacks. Ein Stack ist in diesem Zusammenhang eine Ansammlung an bestimmten Techniken, die zur Webentwicklung verwendet werden. Abschließend folgt im Abschnitt 4.3.3 die Begründung für die Wahl des FARM-Stacks.

4.3.1 ReactJs mit React-Bootstrap

React ist ein JavaScript-Bibliothek für die Erstellung von Benutzeroberflächen.
Bei Bootstrap handelt es sich um ein *CSS*-Framework. *CSS* steht für „Cascading Style Sheet“ und wird verwendet um das Layout einer Website zu formatieren.[8]
Damit Bootstrap in der React-Anwendung hinzugefügt werden kann, wird ein React-Bootstrap-Package wie das „react-bootstrap“ benötigt. [9]

4.3.2 Vergleich von MERN- und FARM-Stack

Für die Entwicklung einer Webanwendung wurden mehrere Webframeworks analysiert und miteinander verglichen. Da die Datenbank mittels *MongoDB* umgesetzt wird, ist eine Voraussetzung an das Webframework, dass dieses *MongoDB* ebenfalls unterstützt. Daher werden im Folgenden der MERN- und der FARM-Stack genauer betrachtet und miteinander verglichen.

MERN-Stack

Der MERN-Stack besteht aus **M**ongoDB, **E**xpressJs, **R**eactJs, **N**odeJs. Die vier Komponenten bilden dabei den Stack.

- **MongoDB** - Dokument-Datenbank
- **ExpressJs** - Node.js-Webframework
- **ReactJS** - Clientseitiges JavaScript-Framework
- **NodeJs** - JavaScript-Webserver

Mit der MERN-Architektur wird eine dreistufige Architektur vollständig mit JavaScript und JSON erzeugt. Diese besteht aus einem Frontend, einem Backend und der Datenbank.

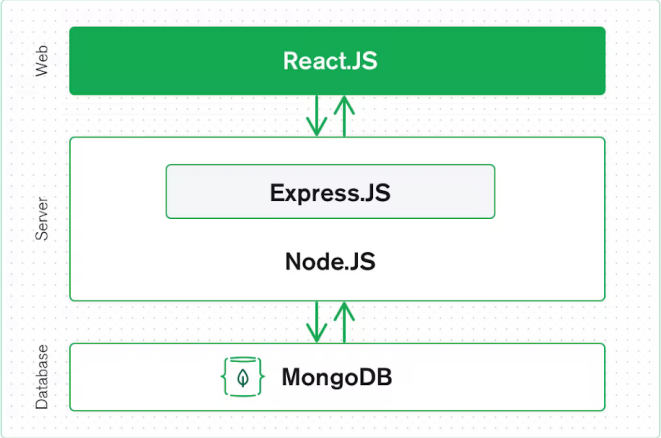

Abbildung 4.13: Darstellung des MERN-Stacks [10]

In der Abbildung 4.13 werden die verschiedenen Ebenen der Architektur graphisch dargestellt. Hierbei ist die oberste Ebene das Frontend, bei dem ReactJs verwendet wird. Die mittlere Schicht besteht aus dem serverseitigen ExpressJs-Framework, welches innerhalb des NodeJs-Servers läuft. Die untere Schicht repräsentiert die Datenbank-Ebene, die durch *MongoDB* realisiert wird[10].

FARM-Stack

Der FARM-Stack besteht genauso wie der MERN-Stack aus den Kompetenten *MongoDB* und ReactJs. Jedoch wird anstelle von Express und Nodejs in dem Fall FastAPI für die Server-Ebene verwendet.

FastAPI ist ein modernes, leistungsfähiges Webframework in Python 3.6+ zur Erstellung von APIs. Es unterstützt auch NoSQL-Datenbanken.[11]
Da im Konzept *MongoDB* verwendet wird und dies vom Framework unterstützt wird, wäre dieses prinzipiell anwendbar.

4.3.3 Entscheidung

Aus Gründen der Vollständigkeit sei erwähnt, dass das bekannte Python-Framework „Django“ während der Konzeptionierung zunächst auch untersucht wurde. Da dieses jedoch offiziell NoSQL-Datenbanken nicht unterstützt und somit eine Nutzung mit *MongoDB* nicht empfohlen wird, wurde es nicht weiter verfolgt.

Die Umsetzung der Software ist prinzipiell sowohl mit dem FARM als auch mit dem MERN-Stack möglich. Beide Webframeworks unterstützen NoSQL-Datenbanken, was auf Grund der Verwendung von *MongoDB* eine Voraussetzung an das Webframework war. Die Entscheidung fiel am Ende zugunsten des FARM-Stacks. Ein entscheidender Aspekt im Entscheidungsprozess war, dass im *ZEA-2* vermehrt Python benutzt wird. Somit ist die Wartbarkeit und Weiterentwicklung auch in der nahen Zukunft durch andere Mitarbeitende gewährleistet. Des weiteren wird auch die erwähnte zweite Bachelorarbeit, die sich mit der universellen Datenschnittstelle für die Ansteuerung von Labormessgeräten mittels SCPI beschäftigt, in Python implementiert. Deshalb könnte die Verwendung von FastAPI die spätere Anbindung an diese erleichtern.

5 Realisierung der Hauptfunktionalität

In diesem Kapitel wird auf die Realisierung, die in dieser Arbeit vorgenommen wurde, eingegangen. Zuerst wird dabei auf die Datenbank eingegangen, wie diese grundlegend aufgebaut ist und wie diese durch *MongoDB* erstellt wird.
Anschließend folgt im Unterkapitel 5.2 eine Erläuterung, wie das Grundgerüst der Oberfläche umgesetzt wurde. Um die Kommunikation zwischen der Oberfläche und der Datenbank nachvollziehen zu können, wird im Unterkapitel 5.3 auf dies näher eingegangen. Genauer gesagt wird die Kommunikation zwischen Datenbank und FastAPI und der Kommunikation von FastAPI zu ReactJs behandelt.
In dem Unterkapitel 5.4 wird auf die Umsetzung weiterer Funktionalitäten eingegangen. Dabei wird beschrieben wie der QR-Code eingebunden und erstellt werden kann. Ebenfalls wird im Abschnitt 5.4.2 erklärt, wie genau der ToggleButton in der Weboberfläche funktioniert, der dem Benutzer die Möglichkeit gibt, zwischen deutsch und englisch zu wählen.

5.1 Realisierung der Datenbank

In diesem Unterkapitel 3.1 wird drauf eingegangen wie für ausgewählte Kategorien die Struktur der Datenbank erstellt wird. Im folgenden Unterkapitel wird erläutert wie wie *Collections* angelegt werden können mit Hilfe von *Schema-Validatoren.*

5.1.1 Struktur der Datenbank

Auf der Grundlage der Analyse in Unterkapitel 3.1 wurde eine *MongoDB*-Datenbank erstellt. Im ersten Schritt sind noch nicht alle später benötigten Daten angelegt worden. So wird in dieser Datenbank die Grundlage durch die Informationen der Geräte, Kalibrierungsdaten, Firmen und dem Messsetup gelegt.
Zu Beginn wurde analysiert, welche Eigenschaften von mehreren Dokumenten immer wieder gleich abgespeichert werden würden, um Zusammenhänge zu erkennen.

Ein Beispiel hierfür wäre die Eigenschaft der „Gerätebeschreibung“ innerhalb der Geräteinformationen. Unter „Gerätebeschreibung“ wurden in der Analyse alle Daten gefasst, die generelle Informationen von einem Gerät sind. Dies hätte die Auswirkung, dass zwei modellgleiche Geräte die gleichen Informationen speichern würden.
Um dies speichereffizient und konsistent zu gestalten, wurde deshalb die Gerätebeschreibung in eine eigene *Collection* ausgelagert. Und die *Collection* „Device“ erhält dabei eine Referenz, die auf die ID eines Dokuments der *Collection* „DevicesGeneralData“ verweist. Diese *Collection* enthält generelle Informationen zu dem Gerät.
Es gibt jedoch auch den Fall, dass Daten gruppiert werden können, ohne dass dabei eine eigene *Collection* nötig ist. Dies lässt sich an dem Beispiel der Kontaktperson einer Firma erläutern. Eine Kontaktperson hat bestimmte Daten, wie zum Beispiel den Namen, eine E-Mailadresse oder Telefonnummer. Da diese Daten jedoch personenbezogen sind und davon ausgegangen wird, dass es bei verschiedenen Firmen keine identischen Kontaktpersonen gibt, wird dies als *Embedded Document* in der *Collection* „Companies“ realisiert.

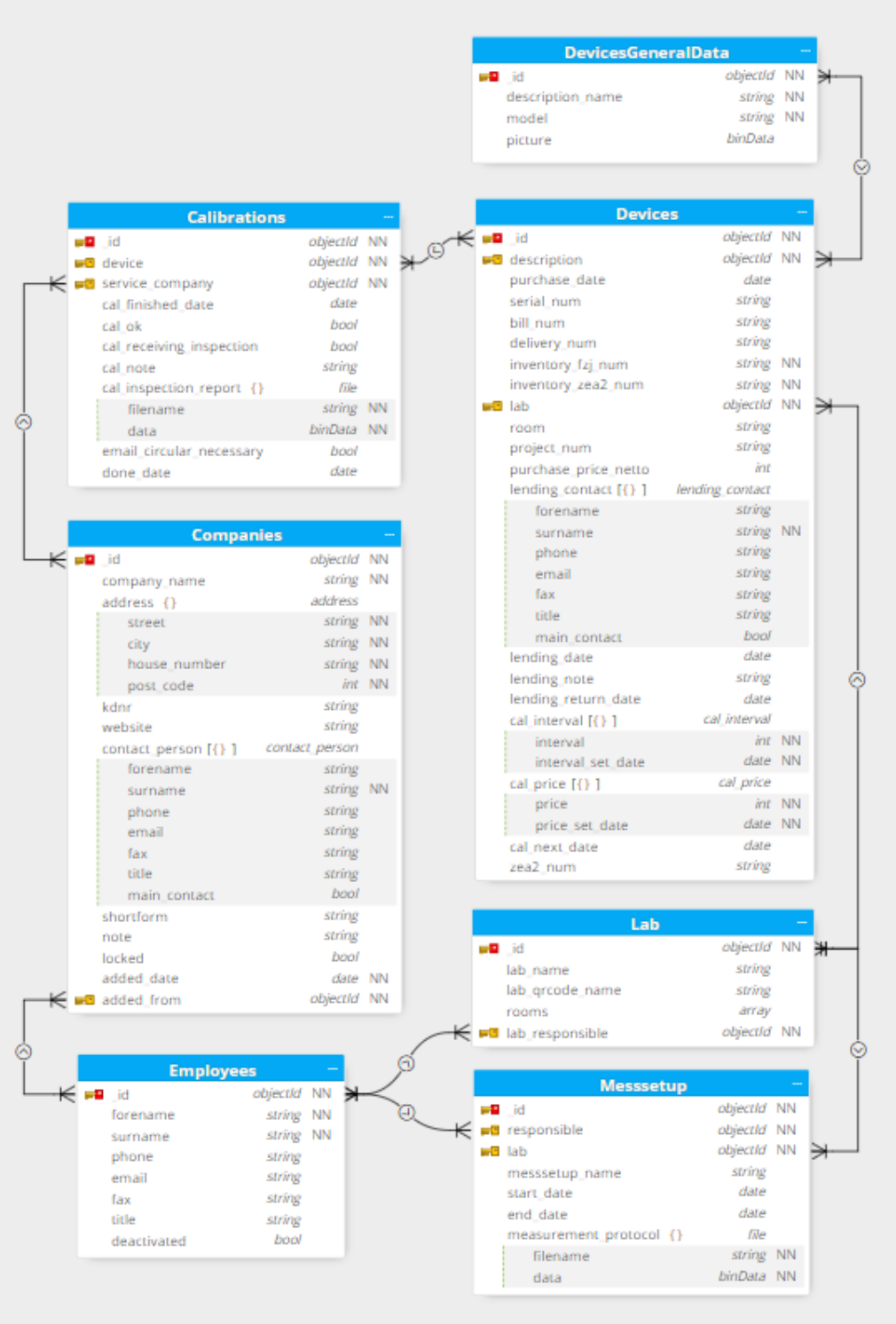

Abbildung 5.1: Diagramm des Datenbankaufbaus

5.1.2 Erstellung einer *Collection* mit Schema-Validatoren

In diesem Abschnitt wird erläutert wie die *Collections* in der Datenbank erstellt werden. Die Interaktion mit der Datenbank wurde mit Python durchgeführt unter Verwendung des Moduls „PyMongo“. Somit können die *Collection*s erstellt werden, indem auf eine Datenbank die Methode „create_collection“ aufgerufen wird. Dieser wird ein Name für die *Collection* übergeben. Die *Collection* kann über eine Angabe einer Schema-Validation „validator“ eine Festlegung von Validierungsregeln erhalten. Der dafür verwendete Quellcode ist im Listing 5.1 auf Seite 44 abgebildet. In Zeile 4 wird zuerst eine Instanz des MongoClient's erstellt. Dazu wird der Klasse „MongoClient“ die Adresse des Datenbankservers als Parameter mit angegeben.

```
1 #Import
2 from pymongo import MongoClient
3
4 client = MongoClient(<Adresse des Datenbankservers>)
5 db = client[<DB>]
6
7 #Methode, die eine neue Collection erstellt
8 db.create_collection(<Name>, validator={<Validator>})
9
```

Listing 5.1: Erstellung einer neuen *Collection*

Auf die Schema-Validation wird im folgenden Abschnitt näher eingegangen.

Schema-Validation

Wie in dem Abschnitt 4.1.2 erläutert wird, ist *MongoDB* ohne spezielle Angaben zunächst schemalos. Durch sogenannte Schema-Validatoren lassen sich jedoch bestimmte Regeln definieren, die vorschreiben, wie Teile der Daten in dem Dokument strukturiert sein sollen. Das Schema der Dokumente wird bei Aktualisierung oder bei dem Einfügen von Daten in die *Collection* durch die definierten Validatoren überprüft. Wie oben erwähnt, werden sie über eine Option „validator“ in der Methode „create_collection“ angegeben. *MongoDB* unterstützt die Validierung durch JSON-Schemas. Dies wird in dem Validator-Ausdruck mit angegeben durch die Verwendung des Operators „$jsonSchema“ .[12]

Der Vorgang der Erzeugung einer *Collection* soll durch ein Beispiel gezeigt werden. Dabei werden die Funktionalitäten von ausgewählten Attributen verdeutlicht.
Der dazugehörige Quellcode ist im Listing 5.2 zu finden und wird im Folgenden näher erläutert. Wie in der Zeile 5 zu sehen, soll die *Collection* Devices erstellt werden.

Notwendige Eigenschaften

In der Validation können durch das Keyword „required“ Attribute angegeben werden, die auf jeden Fall bei einer Erstellung eines Dokuments mit angegeben werden müssen.
In dem gegebenen Beispiel wird, wie in Zeile 14 beschrieben, das Attribut „inventory_fzj_num“ benötigt. Das Attribut steht dabei für die Inventarnummer des Forschungszentrums von dem Gerät. In der eigentlichen Realisierung müssen hier noch weitere Attribute angegeben werden, die aus Gründen der Übersichtlichkeit aber hier ausgelassen werden.

Referenzierung auf eine *Collection*

Wie schon als Beispiel in dem Abschnitt 5.1.1 erwähnt, wurden die generellen Daten eines Gerätes in eine eigene *Collection* gefasst. Damit jedem Gerät ein generelles Modell zugeordnet wird, kann der *Collection* eine ObjektID hinterlegt werden. Diese ObjektID gibt dabei die ID der *Collection* an, auf die referenziert werden soll. Generell kann jedoch ein beliebiges Element angegeben werden. Wenn jedoch auf ein bestimmtes Dokument referenziert werden soll, muss ein *Unique Index* auf dem Wert in der *Collection* existieren.
Im Listing ist dies in Zeilen 16 bis 18 zu sehen und es wird durch das Attribut „description_num“ repräsentiert. Dieser wird eine objektID von der *Collection* übergeben, auf welches generelle Gerätemodell das Gerät verweisen soll.

Realisierung eines Embedded Documents

Da es möglich sein soll, die Historie der Veränderungen in Bezug auf das Kalibrierungsintervall nachvollziehen zu können, müssen hierfür ebenfalls Einstellungen vorgenommen werden. Dabei gibt es zwei Faktoren, die in diesem Zusammenhang eine Rolle spielen. Zum einen das Datum, an welchem das Kalibrierungsintervall gesetzt wurde und zum anderen die Intervallgröße. Da alte Werte weiterhin abgespeichert bleiben müssen, um eine Historie zu ermöglichen, bietet sich in diesem Fall ein Array eines Embedded Documents an. Das Array wird durch die Angabe „array“ im Attribut „bsonType“ angegeben. Ein Embedded Document besteht aus den gleichen Komponenten wie ein normales Dokument. Die benötigten Angaben des Kalibrierungsintervalls „cal_interval“ sind in den Zeilen 27 bis 39 des Listings 5.2 notiert, wobei das beschriebene Array in Zeile 28 definiert wird.

```
#Import
from pymongo import MongoClient

# Name der Collection
dev_devices_coll = "Devices"

client = MongoClient(<HOST>)
db = client[<DB>]

db.create_collection(dev_devices_coll, validator=
{
    "$jsonSchema": {
        "bsonType": "object",
        "required": ["inventory_fzj_num", ... ],
        "properties": {
            "description_num": {
                "bsonType": "objectID"
            },

            ...

            "inventory_fzj_num": {"bsonType": "string"
                                  },

            ...

            "cal_interval": {
                "bsonType": "array",
                "items": {
                    "bsonType": "object",
                    "required": ["interval", "interval_set_date"],
                    "properties": {
                        "interval": {
                            "bsonType": "int"
                        },
                        "interval_set_date": {"bsonType": "date"}
                    }
                }
            }
        }
    }

})
```

Listing 5.2: Erstellung einer neuen *Collection*

Speicherung eines Dokuments

Um Dokumente in der Datenbank abzuspeichern, werden ebenfalls Embedded Documents verwendet. Dazu werden sowohl der Dateiname als auch die binären Daten des Dokuments übergeben. Dies wird im Folgenden verdeutlicht anhand des Attributes „cal_inspection_report“. Dies ist eine Eigenschaft aus der *Collection* für eine Kalibrierung. In diesem Attribut soll das Kalibrierungsdokument abgespeichert werden können. Dazu ist die Angabe des Dateinamens und der binären Daten notwendig. Dies wird im Listing 5.3 in Zeile 3 bis 6 deutlich, da diese Eigenschaften unter „required“ aufgeführt werden.

```
1  "cal_inspection_report": {
2                       "bsonType": 'object',
3                       "required": [
4                           'filename',
5                           'data'
6                       ],
7                       "properties": {
8                           "filename": {
9                               "bsonType": 'string'
10                          },
11                          "data": {
12                              "bsonType": 'binData'
13                          }
14                      }
15                  }
16
```

Listing 5.3: Erstellung einer neuen *Collection*

5.2 Umsetzung der Benutzeroberfläche

Im Folgenden wird das Grundgerüst der Weboberfläche vorgestellt und wie dieses realisiert wurde. In der Umsetzung der Benutzeroberflächen wurde ReactJS mit React-Bootstrap verwendet. Das Grundgerüst besteht dabei aus drei Komponenten und dient dem Proof of Concept. Die dafür benötigten Komponenten sind eine Kopfzeile, eine Sidebar und einer Ansicht, in der über eine Tabelle Daten aus der Datenbank angezeigt werden. Auf alle drei Komponenten wird im Folgenden eingegangen.

5.2.1 Kopfzeile

Wie in der Konzeptionierung in Abschnitt 4.2.1 auf Seite 25 beschrieben, besteht die Kopfzeile aus drei Elementen. Wie in der Abbildung 5.2 zu sehen, wird links durch das Symbol mit den drei Linien die Sidebar realisiert.

Abbildung 5.2: Kopfzeile in der Weboberfläche

Daneben wird durch ein „Dropdown“-Element das Profil realisiert. Im Text steht der Name des eingeloggten Nutzers - hier „Klara Schnorrenberg“. Durch Klicken auf die Komponente wird es zukünftig möglich sein, dass der Benutzer seine Profilansicht ändern kann. Mit Profilansicht ist dabei gemeint, dass der Benutzer von der Labor-Ansicht zu der Qualitätsmanagement-Ansicht wechseln kann. Zudem soll der Benutzer sich darüber wieder abmelden können. Rechts von dem Profilmenü wird dem Benutzer durch einen ToogleButton ermöglicht, zwischen der deutschen und der englischen Sprache zu wechseln. Auf die Realisierung der bilingualen Weboberfläche durch diesen ToggleButton wird im Abschnitt 5.4.2 ab Seite 62 eingegangen.

5.2.2 Realisierung der Sidebar

Die Sidebar wurde implementiert, indem in einer Datei die Daten als Menüelemente abgespeichert wurden. Dazu wurde ein Titel des Elements festgelegt. Bei diesem wurde außerdem die Festlegung getroffen, dass genau dieser Titel ein Key-Element für die spätere Übersetzung des Elements ist. Wie die Übersetzung genau funktioniert, wird im Abschnitt 5.4.2 ab Seite 62 erklärt.
Die Sidebar, wie sie ein Anwender sieht, ist in Abbildung 5.3 dargestellt. Ein Ausschnitt des Codes dazu ist im Listing 5.4 zu finden. In Zeile 5 wird der Titel für das

Element des Labors gesetzt. Außerdem wird über das Element *path* eine Pfadangebe festgelegt, auf welche dieses Element verlinkt werden soll.
In Zeile 7 wird mit dem Element *cName* der Klassenname dieses Elements auf „nav-text“ gesetzt.
Alle Elemente werden dabei durch ein Komma getrennt voneinander aufgelistet.

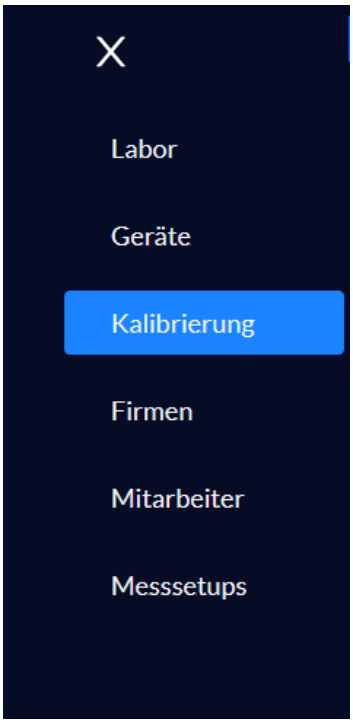

Abbildung 5.3: Sidebar in der Weboberfläche

```
1  import React from "react";
2
3  export const SidebarData = [
4    {
5      title: "lab",
6      path: "/",
7      cName: "nav-text",
8    },
9
10     ...
11
12     {
13      title: "calibration",
14      path: "/calibration",
15      cName: "nav-text",
16   }
17 ];
```

Listing 5.4: Dokument der Sidebar-Daten

Das gerade vorgestellte Dokument speichert die Daten der Sidebar. Um diese Daten einzubinden wird über jedes Element in der Sidebar iteriert und eine *Link*-Komponente erstellt, die auf den Pfad des Elements verwiesen wird.

5.2.3 Implementierung der Kalibrierungsoberfläche

In diesem Unterkapitel wird auf die Implementierung des Grundgerüstes für die Kalibrierungsoberfläche eingegangen. Über das Klicken auf ein Element in der Sidebar wird durch eine „path“ Angabe in dem Element diese Oberfläche angezeigt. Wenn auf „Kalibrierung“ gedrückt wird, wird der Pfad „/calibration“ aufgerufen, wie dem Listing 5.4 zu sehen ist. Die dadurch resultierende Anzeige ist in Abbildung 5.4 als Ausschnitt dargestellt.

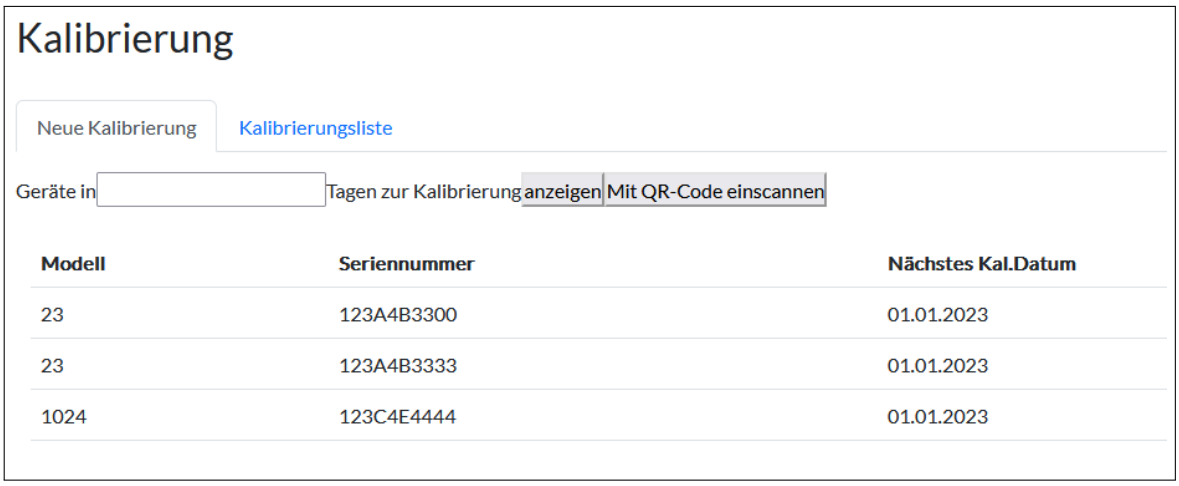

Abbildung 5.4: Grundgerüst der Weboberfläche für die Kalibrierungsinformationen

In der Abbildung ist erkennbar, dass verschiedene Komponenten eingebaut wurden. Eine Überschrift gibt momentan an, welche Oberfläche dargestellt wird. Die Überschrift kann durch einen Heading-Tag angegeben werden. Im Code wurde dies in Form von „<h2>ÜBERSCHRIFT</h2>“ angegeben. Dabei wird dies durch *JSX* ermöglicht. *JSX* steht für JavaScript XML und ist eine Syntaxerweiterung für JavaScript. Es ermöglicht, HTML-Elemente in JavaScript zu schreiben. *JSX* wandelt die HTML-Tags in React-Elemente um [13].
Unter der Überschrift wurden zwei Tabs eingebaut, um innerhalb der Oberfläche die Fenster wechseln zu können. Dazu wurde, wie im Listing 5.5 in Zeile 13 zusehen ist, eine *Tabs*-Komponente verwendet. Diese wurde in der Zeile 3 importiert. Die *Tabs* bestehen dabei aus einzelnen *Tab*-Elementen. In der Oberfläche werden zwei Tabs verwendet. Daher wird ein *Tab* für die Ansicht benutzt, über die eine neue Kalibrierung erstellt werden kann, und ein weiterer für eine Auflistung der Kalibrierungen.
Dies wird im Listing 5.5 in Zeile 18 bis 20 für den einen *Tab* umgesetzt und in Zeile

21 bis 23 für den anderen *Tab*. Jeder *Tab* hat dabei einen *eventKey*. Dieser Wert des *eventKey* kann genutzt werden, um wie in Zeile 14 zu sehen einen *Tab* auf *default* zu setzen, sodass dieser beim Öffnen der Oberfläche automatisch geöffnet ist.
Wie das in der Zeile 11 stehende *useTranslation()* benutzt wird, wird im Abschnitt 5.4.2 ab Seite 62 erläutert. Dies ist für die Umsetzung der mehrsprachigen Benutzeroberfläche nötig.
Innerhalb des *Tab*-Tags werden die Komponenten geschrieben, die in diesem Fenster angezeigt werden sollen. Für eine bessere Übersicht wurde das im Beispiel des Listings in separate Dateien ausgelagert.
In dem Beispiel ist dies durch die Verwendung des Imports aus Zeile 6 innerhalb des *Tabs* in Zeile 19 zu erkennen.

```
1  //Import
2  import Tab from 'react-bootstrap/Tab';
3  import Tabs from 'react-bootstrap/Tabs';
4  import { useTranslation } from 'react-i18next'
5
6  import NewCalibration from "./NewCalibration/NewCalibration";
7  import AddNewCalibration from "./AddNewCalibration/AddNewCalibration";
8
9
10 function TabsHorizontal_Calibration() {
11   const {t}= useTranslation()
12   return (
13     <Tabs
14       defaultActiveKey="calibration_new"
15       id="uncontrolled-tab-example"
16       className="mb-3"
17     >
18       <Tab eventKey="calibration_new" title={t("calibration_new")}>
19             <NewCalibration/>
20       </Tab>
21       <Tab eventKey="calibrationlist" title={t("calibrationlist")}>
22             <AddNewCalibration/>
23       </Tab>
24     </Tabs>
25   );
26 }
27
28 export default TabsHorizontal_Calibration;
```

Listing 5.5: Horizontale Tabbar

Im Folgenden wird nun näher auf diese Umsetzung aus der gerade genannten eingebundenen Datei für einen neuen Kalibrierungsantrag eingegangen.

5.2.4 Erstellung eines neuen Kalibrierungsantrages

Die Komponenten der Oberfläche, auf die dieser Abschnitt eingeht, werden in Abbildung 5.5 dargestellt.

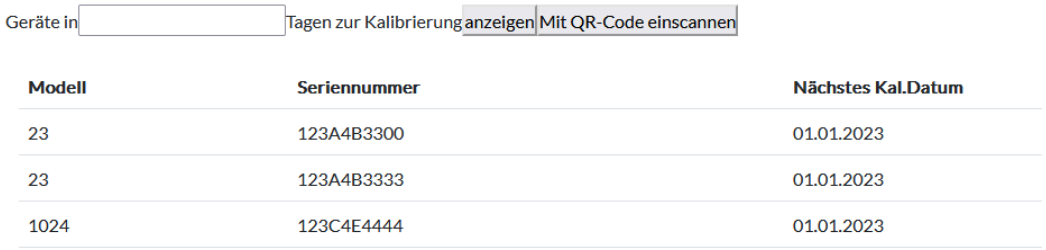

Abbildung 5.5: Grundgerüst des Fensters für eine neue Kalibrierung

Diese angezeigten Daten sind Komponenten aus zwei *Collections*. Mit Modell wird eine generelle Modellnummer definiert, zu dem wird gerätespezifisch die Seriennummer und das nächste Kalibrierungsdatum angegebn. In der späteren Umsetzung werden noch weitere Eigenschaften angezeigt aus den beiden *Collections*. Für das „Proof of concept“ wurde sich auf ein Minimum beschränkt.
In der oberen Zeile wurde mittels Labels die Textanzeige erzeugt. Das Feld, in dem der Benutzer die Tage eintragen kann, wurde durch eine *Input*-Komponente erzeugt. Ebenfalls wurden zwei Buttons integriert. Damit ein besserer Überblick über die darunter liegende Tabelle gegeben werden kann, wurden diese Komponenten in dem Listing 5.6 nicht aufgeführt.

```
1  //Import
2  import React from "react";
3  import NewCal_Table from "./NewCalibration_Table";
4  import {useEffect, useState} from "react";
5  import axios from "axios";
6
7  function NewCalibration() {
8      const [calDevicesList, setCalDevicesList] = useState([{}])
9
10     useEffect(() => {
11       axios.get('http://localhost:8000/api/calDeviceList')
12         .then(res => {
13             setCalDevicesList(res.data)
14         })
15     }, []);
16
17     return (
18         <div>
19
20             ...
21
22             <NewCal_Table calDevicesList={calDevicesList} />
23         </div>
24     );
25 }
26 export default NewCalibration;
```

Listing 5.6: Grundgerüst des Fensters für das Hinzufügen einer neuen Kalibrierung

Die Tabelle wird in einem separaten Dokument erstellt und in das Fenster eingebunden. Dies wird, wie im vorherigen Abschnitt an einem anderen Beispiel gezeigt, über einen Import möglich. Dann kann dieser Import als eine *Tag*-Komponente verwendet werden. Dies ist in der Zeile 22 kodiert.
Die Daten, die in der Tabelle angezeigt werden sollen, werden über eine Liste *calDevicesList* übergeben. Über die Funktion *useEffekt* in Zeile 10 wird diese Liste mit Werten gefühlt, indem über eine *Get*-Anfrage die Daten von der Serverseite angefragt werden. Wie genau die Daten aus der Datenbank gelesen werden und an die Benutzeroberfläche gelangen wird im Unterkapitel 5.3 erläutert.
Der Funktion der Tabelle werden die Daten übergeben. Dieser Übergabeparamter wird im Listing 5.7 in Zeile 5 als *data* deklariert. Es handelt sich dabei um ein JavaScript Objekt.
Das Objekt kann mehrere *Properties* besitzen. Dabei werden die Werte des Objekts als *name:value*-Paar formuliert und der Name mit einem Doppelpunkt vom Wert getrennt. [14]
In dem Beispiel werden innerhalb des Objekts *data* mit dem Namen *calDevicesList* die Werte in einem Array zugewiesen.
Die Tabelle ist hier so aufgebaut, dass eine Tabellen-Komponente erstellt wird, was in dem Listing ab Zeile 9 zu erkennen ist. Die Tabellen-Komponente besteht dabei

aus einem Teil für die Kopfzeile der Tabelle und einem Teil für die restlichen Zellen der Tabelle.
Die Überschriften der Tabellenspalten werden mittels der Komponente *thead* gesetzt. In dieser wird mittels *<tr>* eine Zeile definiert und dann, wie in Zeile 12 zu sehen ist, über das Element <th> die Überschriften gesetzt. Da diese übersetzt werden sollen je nach angezeigter Sprache, wurde erneut die t-Funktion verwendet, die im Abschnitt 5.4.2 ab Seite 62 erläutert wird.
Von Zeile 17 bis 25 werden die Daten in die Tabelle geschrieben. Dies wurde mittels eines *tbody*-Elements erreicht. Dieses fasst Tabellenzeilen zu einer Gruppe zusammen, die Tabellendaten enthalten. In dieser wird mittels der *map*-Funktion über das Array iteriert, welches die Daten enthält. Für jede Zeile wird in diesem Fall über das Element *<td>* der ausgelesene Wert in die Zelle geschrieben.
In Zeile 21 des Listings ist dies für die Zelle der Seriennummer zu sehen.

```
1  //Import
2  import React from "react";
3  import { useTranslation } from "react-i18next";
4
5  function NewCalibration_Table(data) {
6      const { t } = useTranslation();
7      return (
8          <div>
9              <table className="table">
10                 <thead>
11                     <tr>
12                         <th scope="col">{t("model")}l</th>
13                         <th scope="col">{t("serial_num")}</th>
14                         <th scope="col">{t("cal_next_date")}</th>
15                     </tr>
16                 </thead>
17                 <tbody>
18                     {data.calDevicesList.map((device) => (
19                         <tr>
20                             <td>{device.model}</td>
21                             <td>{device.serial_num}</td>
22                             <td>{device.cal_next_date}</td>
23                         </tr>
24                     ))}
25                 </tbody>
26             </table>
27         </div>
28     );
29 }
30
31 export default NewCalibration_Table;
```

Listing 5.7: Tabellen-Komponente

5.3 Kommunikation zwischen Benutzeroberfläche und der Datenbank

Im letzten Kapitel wurde die Umsetzung bestimmter Komponenten der Weboberfläche vorgestellt. Anhand der erläuterten Tabelle im Abschnitt 5.2.4 wird nun der Prozess gezeigt, wie die Daten aus der Datenbank zur Oberfläche gelangen.
Die Datenbank und die Benutzeroberfläche kommunizieren mittels FastAPI. Sobald eine Anfrage von der Benutzeroberfläche gestellt wird, werden diese durch FastAPI verarbeitet und die entsprechenden Daten aus der Datenbank gelesen. Diese werden dann durch FastAPI an das Frontend zurückgegeben.

5.3.1 ReactJs und FastAPI

Damit Daten abgerufen werden können, werden *Get*-Anfragen gestellt. Im Listing 5.8 ist ein Code-Ausschnitt des zuvor verwendeten Listings 5.6 auf Seite 51 dargestellt. Damit die Anfrage durchgeführt wird, wenn die Komponente eingebunden wird, wird die Methode „useEffect“ in Zeile 3 aufgerufen.

```
1 const [calDevicesList, setCalDevicesList] = useState([{}])
2
3 useEffect(() => {
4   axios.get('http://localhost:8000/api/calDeviceList')
5     .then(res => {
6         setCalDevicesList(res.data)
7     })
8 }, []);
```

Listing 5.8: Get-Anfrage; Ausschnitt des Listings 5.6

In dieser Methode wird unter Verwendung von *Axios* eine *Get*-Abfrage an die Serverseite gestellt. *Axios* ist eine Frontend-HTTP-Bibiliothek.
Axios hat Funktionsnamen, die allen HTTP-Methoden entsprechen. Um eine *Get*-Anfrage durchzuführen, wird die Methode *get()* verwendet. Außerdem benötigt *Axios* nur einen „.then()“-Callback, um auf die angeforderten JSON-Daten zuzugreifen.[15]
Der Methode *get()* wird dabei als Übergabeparameter eine URL aus dem API-Endpunkt übergeben. Da der Backendserver momentan noch lokal auf dem Port 8000 läuft, wird dies wie dem Listing Zeile 4 vollständig angegeben. Dabei wird in dem Fall des Beispiels die Route von „api/calibrationDeviceList“ abgefragt. Dies ist eine festgelegte Route, die speziell die für diese Tabelle angeforderten Werte abfragt. Auf die angeforderten Daten wird dann über ein „.then()“-Callback zugegriffen. Das Attribut „res“ in Zeile 5 repräsentiert dabei JSON-Daten. Auf diese wird wie in Zeile 6

gezeigt durch den Aufruf von „res.data“ zugegriffen.
In dem Beispiel werden die Daten dazu verwendet, um mittels der Methode „setCalDevicesList“ die List „calDevicesList“ zu aktualisieren. Diese Liste wird dann, wie in dem Abschnitt 5.2.4 ab Seite 50 beschrieben, verwendet, um die Daten in die Tabelle zu schreiben.

Im folgenden Unterkapitel wird auf Erläuterung eingegangen, was bei einer Get-Anfrage auf der Serverseite passiert.

5.3.2 FastAPI und Datenbank

Um nachvollziehen zu können, wie eine *Get*-Anfrage auf der Serverseite bearbeitet wird, wird vorab auf FastAPI eingegangen. Dazu wird anschließend erläutert wie speziell die *Get*-Anfrage aus dem Beispiel des vorherigen Abschnitts von der Serverseite verarbeitet wird.

Main-Datei des FastAPI-Servers

Durch die Main-Datei wird mittels *Uvicorn* ein Webserver gestartet.
Uvicorn ist eine „ASGI“ Webserver-Implementation in Python. Dabei steht „ASGI“ für „Asynchronous Server Gateway Interface“. Dies stellt sowohl ein Standard für asynchrone als auch für synchrone Anwendungen bereit.[16] Dabei wird ein „Host“ und ein „Port“ in der *run*-Methode angegeben. In dem Listing 5.9 ist dies ab der Zeile 37 dargestellt. Die Werte, die den Attributen dabei zu geordnet werden, wurden in eine separate Konfigurationsdatei ausgelagert.
Eine Instanz der FastAPI-Applikation wird in Zeile 18 erstellt. Dabei wird eine Middleware hinzugefügt. Dies ist dabei wie in Zeile 11 zu sehen eine *CORS-Middleware*. Der Begriff *CORS* steht für „Cross-Origin Resource Sharing“ und bezieht sich auf die Situation, in der sich das Frontend und das Backend in einem anderen „Ursprung“ befinden.[17] Momentan läuft das Backend auf „http:localhost:8000“ und das Frontend unter „http:localhost:3000“.
Ebenfalls wurden EventHandler-Funktionen definiert. Diese sind Funktionen, die ausgeführt werden müssen bevor die Anwendung startet oder wenn die Anwendung beendet wird. Die Funktion „startup_db_client()“ in Zeile 22 wurde dabei durch Zeile 21 als eine EventHandler-Funktion definiert. In dieser wird für die *app* ein Client angelegt und eine Datenbank zugewiesen.

```
1  from fastapi import FastAPI
2  import uvicorn
3  from motor.motor_asyncio import AsyncIOMotorClient
4  from config import settings
5  from starlette.middleware import Middleware
6  from starlette.middleware.cors import CORSMiddleware
7  from routers.CalibrationRouter import get_api_router_cal
8
9  middleware = [
10     Middleware(
11         CORSMiddleware,
12         allow_origins=['*'],
13         allow_credentials=True,
14         allow_methods=['*'],
15         allow_headers=['*']
16     )
17 ]
18 app = FastAPI(middleware=middleware)
19
20
21 @app.on_event("startup")
22 async def startup_db_client():
23     app.mongodb_client = AsyncIOMotorClient(settings.DB_URL)
24     app.mongodb = app.mongodb_client[settings.DB_NAME]
25
26
27 @app.on_event("shutdown")
28 async def shutdown_db_client():
29     app.mongodb_client.close()
30
31 # Method to include the router
32 app.include_router(get_api_router_cal(app), tags=["calDeviceList"])
33
34
35 # main function to run the server
36 if __name__ == "__main__":
37     uvicorn.run(
38         "main:app",
39         host=settings.HOST,
40         reload=settings.DEBUG_MODE,
41         port=settings.PORT,
42     )
```

Listing 5.9: Main-Datei des FastAPI-Servers

Mit dem Aufruf der Methode „include_router“ in Zeile 32 wird der Router in FastAPI eingebunden. Dieser Router wird in der Methode „get_api_router_cal“ erstellt und definierte Routen zugewiesen. Im folgenden Abschnitt wird auf diese Funktion weiter eingegangen.

Umsetzung der Routen

Alle Routen der Kalibrierungsdaten werden innerhalb einer Funktion erstellt und einem Router zugewiesen. Dazu wird in dieser Funktion ein neuer APIRouter erstellt. Die Methode „get_calDeviceList“ soll jedes mal aufgerufen werden, wenn eine „HTTPGet“Anfrage vom Frontend eingeht. Dazu wird dies mittels des Python *Decorators* [18] umgesetzt. Dazu wird wie im Listing in Zeile 9 zu erkennen, über die Funktion in diesem Fall „@router.get('/api/calDeviceList')“ definiert.

```
1  from fastapi import APIRouter
2  from database.CalDeviceList import fetch_all_calDeviceList
3
4
5  def get_api_router_cal(app):
6      # Create a Router
7      router = APIRouter()
8
9      @router.get("/api/calDeviceList")
10     async def get_calDeviceList():
11         response = await fetch_all_calDeviceList()
12         return response
13
14     return router
```

Listing 5.10: Datei mit den Routen der Kalibrierung

Innerhalb der Funktion wird, wie in Zeile 11 zu sehen, eine Funktion aufgerufen, die die Daten aus der Datenbank liest. Das Ergebnis dieser Funktion wird an den Client zurückgegeben. Wie genau diese Funktion zum Auslesen aus der Datenbank implementiert wurde, wird im folgenden Abschnitt erläutert.

Datenbankabfrage

Damit die Daten aus der Datenbank gelesen werden können, wird ein *client*-Objekt und mit diesem ein Datenbank-Objekt erstellt. In dem Listing 5.11 ist dies in Zeile 5 und 6 dargestellt.
In dem Beispiel stehen die Daten zu dem Modell in einer anderen *Collection* als die restlichen Daten. Um die benötigten Daten aus beiden *Collections* zu erhalten, wird die Methode „aggregate()“ aufgerufen. Die Methode verarbeitet mehrere Dokumente und gibt die zusammengesetzten Daten zurück, da eine sogenannte Pipeline der Methode mit übergeben wird. Diese wird im folgenden Abschnitt erläutert.
Die Methode gibt dabei einen Cursor zurück, der auf das Dokument zeigt, welches in der letzten Phase der Aggregationspipeline-Operation erzeugt wurde. Damit eine Liste mit den Daten des Modells, der Seriennummer und dem nächsten Kalibrierungsdatum zurückgegeben wird, werden diese explizit für jedes Dokument in dem Cursor

ausgelesen und als Element in der Liste gespeichert. Dies wird in dem Listing in der for-Schleife von Zeile 12 bis 17 verdeutlicht.

```
1  from database.pipelines import calibration_device_list
2  import pymongo as pm
3
4  client = pm.MongoClient("zel1026")
5  database = client.Laborequipment
6
7
8  async def fetch_all_calDeviceList():
9      calibrationDeviceList = []
10     collection_calibration = database.Devices
11     cursor = collection_calibration.aggregate(calibration_device_list)
12     for document in cursor:
13         element = {
14             "serial_num": document['serial_num'],
15             "cal_next_date": document['cal_next_date'].strftime('%d.%m.%
       ↪ Y'),
16             "model": document['result']['model']
17         }
18         calibrationDeviceList.append(element)
19     return calibrationDeviceList
```

Listing 5.11: Auslesen von Daten aus der Datenbank

Pipeline

Eine Pipeline besteht aus einer oder mehreren Phasen, in denen Dokumente verarbeitet werden. Jede Phase führt eine Operation an dem Eingabedokument durch. In einer Phase ist es möglich, dass das Dokument gefiltert, gruppiert oder bestimmte Werte berechnet werden. Die Dokumente, die durch eine Phase ausgegeben werden, werden an die nächste Stufe weitergeleitet. Die Pipeline kann dabei mehrere Dokumente zusammenfassen. [19]

Wie in dem vorherigen Abschnitt erwähnt, sind in dem Beispiel die Daten für das Modell in einer anderen *Collection* als die restlichen Daten. Die Modell-Bezeichnung ist in der *Collection* „DevicesGeneralData“, die alle generellen Daten zu einem Gerät abspeichert. Die Daten für die Seriennummer als auch für das nächste Kalibrierungsdatum sind in den Dokumenten der *Collection* „Devices“ abgespeichert und haben ebenfalls eine ObjektID. Diese ObjektID gibt an, auf welches Dokument aus der *Collection* „DevicesGeneralData“ gezeigt wird. Damit wird jedem Gerät ein Dokument zugeordnet, in dem generelle Informationen zu diesem Gerät stehen. Über diese Angabe der ObjektID lassen sich beide *Collection*s miteinander verbinden.
Dies wird über drei Phasen der Pipeline realisiert.

```
calibration_device_list = [
    {
        '$lookup': {
            'from': 'DevicesGeneralData',
            'localField': 'description_num.$id',
            'foreignField': '_id',
            'as': 'result'
        }
    }, {
        '$unwind': {
            'path': '$result',
            'preserveNullAndEmptyArrays': True
        }
    }, {
        '$project': {
            'result.model': 1,
            'serial_num': 1,
            'cal_next_date': 1,
            '_id': 0
        }
    }
]
```

Listing 5.12: Pipline für die Kalibrierungsliste

Operation „$lookup“

Damit die Dokumente verbunden werden, wird in der ersten Phase der Pipeline dies durch die Operation „$lookup“ durchgeführt. Dabei hat der Operator eine definierte Syntax, welche in dem Listing 5.13 aufgeführt ist.

```
{
   $lookup:
     {
      from: <collection to join>,
      localField: <field from the input documents>,
      foreignField: <field from the documents of the "from" collection>,
      as: <output array field>
     }
}
```

Listing 5.13: Syntax von $lookup

Mittels des Attributs „from“ in Zeile 4 wird die *Collection* angegeben mit der die *Collection*, auf welcher die Pipeline aufgerufen wurde, verbunden werden soll.
In dem Beispiel wird die „aggregate“-Methode auf der *Collection* „Device“ aufgerufen. Diese soll mit der *Collection* „DevicesGeneralData“ verbunden werden. Daher wird, wie in Zeile 4 des Listings 5.12 zu erkennen, die *Collection* „DevicesGeneralData“ dem Attribut „from“ zugewiesen. Zu beiden *Collection*s, die miteinander verbunden werden sollen, wird ein Attribut des Dokuments angegeben.

Jedem Gerät wird ein eindeutiges Dokument der „DevicesGeneralData“-*Collection* zugewiesen, da die Dokumente der Geräte eindeutige ObjektIDs der „DevicesGeneralData“-*Collection* gespeichert haben. Das Feld, in der die ObjektID in dem Dokument der „Devices“-*Collection* und die ID des Dokuments der „DevicesGeneralData“-*Collection* gespeichert wird, werden als Verbindungsfelder betrachtet. Diese Angabe wurde in dem Listing in Zeile 5 und 6 gemacht.
Mit dem Attribut „as“ wird ein Name des neuen Arrays angegeben. Das Array enthält dabei alle passenden Dokumente. Da in dem Beispiel ein Primärschlüssel als Verbindung der *Collection* verwendet wurde, wird in dem Array nur ein Element enthalten sein.
Da das Ergebnis von einer „$lookup“-Operation ein Array ist und für das Auslesen einzelne Dokumente benötigt werden, wird in der zweiten Phase der Operator „$unwind“ aufgerufen.

Operation „$unwind“
Der Operator „$unwind“ dekonstruiert das Array aus dem Eingabedokument. Für jedes Element aus dem Array wird ein eigenes Dokument ausgegeben. Dazu wird über das Attribut „path“ das Array angegeben. In Zeile 12 wird das Attribut „preserveNullAndEmptyArrays“ auf True gesetzt. Hiermit wird erreicht, dass das Dokument auch ausgegeben wird von der Operation „$unwind“, selbst wenn der Pfad null ist, fehlt oder ein leeres Array betrachtet wird.
Das Ergebnisdokument erhält eine ID. Diese ist für die Ausgabe in der Tabelle nicht notwendig. Aus diesem Grund wird in einer dritten Phase die Operation „$project“ aufgerufen.

Operation „$project“
Die „$project“-Operation gibt die Dokumente mit den angeforderten Feldern zurück. Dabei können die Felder, wie in dem Listing in Zeile 16 bis 19 zu sehen, durch eine Zuweisung von „0“ oder „1“ angegeben werden. In diesem Fall steht die Eins dafür, dass dieses Attribut in dem neuen Dokument enthalten sein soll und bei einer Angabe einer Null wird dieses Attribut in dem neuen Dokument nicht gespeichert.

CRUD-Operationen

Die *CRUD*-Operationen sind eine Bezeichnung für die vier wichtigsten Operationen für Datenbankanwendungen. Die Operationen haben dabei folgende Bedeutung:

- Create: Anlegen eines Datensatzes
- Read : Lesen eines Datensatz
- Update: Aktualisieren des Datensatzes
- Delete: Löschen des Datensatzes

Zu jeder CRUD-Operation gibt es eine passende HTTP-Methode. In Tabelle 5.1 sind die Zuordnungen zu sehen.

Tabelle 5.1: Gegenüberstellung von CRUD-Operatoren und HTTP-Methoden

CRUD	**HTTP**
CREATE	POST/PUT
READ	GET
UPDATE	PUT/POST/PATCH
DELETE	DELETE

In diesem Unterkapitel wurde ein „proof of concept“ erfolgreich mittels der „GET“-Anfrage durchgeführt. Es wurde gezeigt, wie bestimmte Elemente in der Oberfläche angezeigt werden. Anhand einer „Get“-Anfrage wurde die Kommunikation zwischen ReactJs, FastAPI und der Datenbank untersucht. Dies bietet eine Grundlage, auf der das Projekt umgesetzt werden kann. Die anderen CRUD-Operationen wurden anhand von Beispielen getestet.

5.4 Umsetzung weiterer Funktionalitäten

Innerhalb dieses Unterkapitels werden zwei allgemeingültige Funktionalitäten der Software umgesetzt. Zuerst wird dafür auf die Generation der Excel-Datei zur Erstellung der QR-Codes eingegangen. Anschließend folgt im Abschnitt 5.4.2 die Grundidee zur Umsetzung einer bilingualen Weboberfläche.

5.4.1 Excel-Generator der QR-Codes

In der zu entwickelnden Software soll es möglich sein, Geräte mittels eines QR-Code-Scanners zu einem Messsetup hinzuzufügen. Um die QR-Codes verwenden zu können, sollen sie ebenfalls über die Anwendung erstellt werden können. Dazu ist es notwendig, wie in der Aufgabenstellung im Kapitel 2 erläutert, zu ausgewählten Geräten eine Excel-Tabelle zu generieren. Mit dieser Excel-Tabelle erzeugt das Tool des *Brother*-Druckers anschließend die QR-Codes.
Die Auswahl der Geräte erfolgt vom Benutzer über die grafische Oberfläche. Die Software erstellt dann eine Liste der ausgewählten Geräte mit den benötigen Informationen für den QR-Code. Benötigt werden die folgenden Informationen, wobei in den Klammern die Namen der Attribute aus der Datenbank zu finden sind: Seriennummer (zugehöriges Attribut in der DB: *serial_num*), die Inventarnummer des Forschungszentrums (*inventory_num*), das Datum der nächsten Kalibrierung (*cal_next_date*) und die interne Nummer des *ZEA-2*s (*zea2_num*). In welchem Labor das Gerät steht (*lab_qrcode_acronym*), wird ebenfalls mit einem Kürzel angegeben.
Diese Liste wird intern der Methode zur Generierung der Excel-Datei übergeben. In der Methode wird die Liste dann durch iteriert und die Daten an die entsprechende Zelle der Excel-Tabelle geschrieben. Dies wird mithilfe von *xlsxwriter* in Python realisiert.
In der Abbildung 5.6 wird in einem Ausschnitt aus der Excel-Tabelle verdeutlicht, wie diese dargestellt wird.

	A	B	C	D	E	F
1	ID	SN	Cal	Inv	Lab	
2	1	MY52290036	19.07.2025	ZEA2.17.PS.002	GEO Lab	
3	2	MY52290037	20.07.2025	ZEA2.17.PS.003	GEO Lab	
4	3	MY52290038	21.07.2025	ZEA2.17.PS.004	GEO Lab	
5	4	MY52290039	22.07.2025	ZEA2.17.PS.005	GEO Lab	
6	5	MY52290040	23.07.2025	ZEA2.17.PS.006	GEO Lab	
7						
8						

Abbildung 5.6: Ausschnitt der generierten Excel-Datei

Die Daten der Geräte werden zeilenweise angegeben. Zu jedem Gerät wird in Spalte A die ID ausgegeben, gefolgt von der Seriennummer des Geräts(SN). Wichtig sind die Daten in der Spalte C, die angeben bis wann das Gerät als kalibriert angesehen werden kann. In Spalte D und E folgen abschließend die interne ZEA-2 Inventarnummer, sowie die Angabe, in welchem Labor sich das Gerät befindet.

Die erzeugte Excel-Datei kann dann in dem Tool von *Brother* eingelesen werden und damit die QR-Codes erzeugt sowie ausgedruckt werden. In Abbildung 5.7 ist beispielhaft ein solcher Ausdruck abgebildet. Der Aufkleber beinhaltet dabei nicht nur den QR-Code, sondern noch weitere der zuvor beschriebenen Informationen. Alle Daten, die in der Excel-Tabelle aufgelistet sind, werden auch auf dem Aufkleber angezeigt. Im QR-Code wird dabei die Inventarnummer des *ZEA-2*'s abgespeichert. Das bedeutet, wenn der QR-Code eingescannt wird, gibt der Barcode die Inventarnummer des *ZEA-2*'s wie eine Art Tastatureingabe aus.

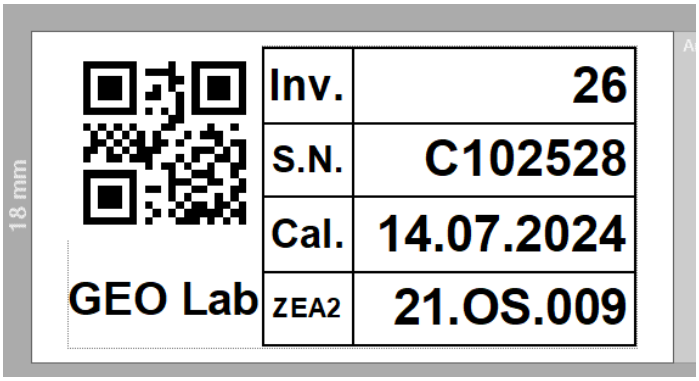

Abbildung 5.7: Beispiel eines Aufklebers mit QR-Code

5.4.2 Bilinguale Weboberfläche

Damit es für den Benutzer die Möglichkeit gibt, zwischen der deutschen und englischen Sprache zu wählen, wurde wie in der Konzeptionierung der Benutzeroberfläche geplant, ein ToggleButton eingebaut. Dieser verändert bei einer Interaktion des Benutzers alle sprachlichen Komponenten auf der Benutzeroberfläche zu der entsprechenden Sprache. Um dies auf der Client-Seite zu realisieren, wurde das *react-i18next*-Framework verwendet. Um die Funktionsweise und Benutzung dieses Frameworks nachvollziehen zu können, wird dies mittels des Beispiels einer Überschrift erläutert. Zuerst wird für jede Sprache eine JSON-Datei angelegt, welche ähnlich wie ein Wörterbuch verwendet wird. Zu jeder sprachlichen Komponente, die in der Oberfläche zu finden ist, wird ein Key hinterlegt. In den JSON-Dateien wird jedem Key eine entsprechende Übersetzung zugeordnet.
Diese Zuordnung wird beispielhaft in Listing5.14 und Listing 5.15 gezeigt. Dem Key „lab“ wird in beiden JSON-Dateien das jeweils anzuzeigende Wort zugewiesen.

```
{
    "lab":"Labor"
}
```

Listing 5.14: deutsche Übersetzungsdatei

```
{
    "lab":"Lab"
}
```

Listing 5.15: englische Übersetzungsdatei

Dabei steht links von einem Doppelpunkt jeweils die Bezeichnung des Keys und nach dem Doppelpunkt die sprachliche Komponente, die unter diesem Key stehen soll.
Der Key wird dann in dem JavaScript-Dokument verwendet, in der die Beispiels-Überschrift implementiert ist. Dazu wird wie in dem Codeausschnitt 5.16 zu sehen dieser Key mithilfe der Übersetzungsfunktion „useTranslation“ verwendet. Dazu wird in der Funktion zu erst ein Objekt dieser Übersetzungsfunktion angelegt, wie in Zeile 6 des Listings zu sehen ist. Mit der „t-Funktion“ kann dann, wie in Zeile 11 dargestellt, eine Sprachkomponente übersetzt werden. Dies wird realisiert in dem die Angabe: t(„<Key>“) mit dem entsprechenden Key beschrieben wird. Diese Methode setzt an diese Stelle die Übersetzung der momentan ausgewählten Sprache.

```
1  #Import
2  import React from 'react';
3  import {useTranslation} from "react-i18next";
4
5  function Lab_View() {
6    const { t } = useTranslation();
7
8    return (
9      <div>
10         # Beispiel fuer eine Uebersetzung der Ueberschrift
11         <h2>{t("lab")}</h2>
12     </div>
13   );
14 }
15
16 export default Lab_View;
```

Listing 5.16: JavaSkript-Datei mit einer Übersetzung

Die Sprache wird in der Oberfläche mit Hilfe des Togglebuttons vom Benutzer ausgewählt. Sobald der Togglebutton verändert wird, wird eine „onChange“-Methode des ToggleButtons aufgerufen. In dieser wird dann von *i18next* die Methode „changeLanguage“ aufgerufen. Der Methode wird dabei in einem vorher definierten Code die entsprechende Sprache übergeben. Die JavaScript-Datei für den ToggleButton besitzt dabei eine Liste „languages“. In dieser ist für beide Sprachen ein Name hinterlegt, wie diese auf der Oberfläche angezeigt werden soll. Außerdem wird für jede Sprache der zuvor beschriebene Code hinterlegt.

Der gleiche Code wird in der Initialisierung von *i18next* verwendet. Die Initialisierung findet in der „Index.js“ statt. Dort wird *i18next* angelegt. In der „.init“ wird ein Array „supportedLngs“ angegeben, in dem diese Codes aufgelistet werden.

In dem Listing 5.17 wird dies in Zeile 4 gezeigt. Mittels der Attributangabe für „loadPath“ wird in Zeile 7 der Pfad zu den JSON-Dateien angegeben. Dies wird je nach betrachteter Sprache angepasst.

```
1 i18next
2   ...
3   .init({
4     supportedLngs: ['en', 'de'],
5     ...
6     backend: {
7       loadPath: '/assets/locales/{{lng}}/translation.json',
8     }
9   })
```

Listing 5.17: Code-Ausschnitt für das Initialisiern von i18next

6 Zusammenfassung

Im Zuge dieser Bachelorarbeit sollte ein Konzept für die Umsetzung einer *Datenmanagementsoftware* entwickelt werden. Dazu wird einleitend an Hand des bisherigen Vorgehens die Motivation einer *Datenmanagementsoftware* erläutert. Das Ziel dieser *Datenmanagementsoftware* ist es, den Laborbetrieb im *ZEA-2* zu verbessern.

Hierfür wird zunächst das Gesamtprojekt vorgestellt und die Anforderungen an die *Datenmanagementsoftware* analysiert und definiert. Des Weiteren werden die Kategorien der Informationen erklärt. Um die Daten zu speichern, wird das dokumentbasierte Datenbanksystem *MongoDB* verwendet werden.

Zusätzlich werden Benutzeroberflächen geplant und mittels Qt-Creator designt. Da nicht fest definiert ist, ob die Software als Desktop oder als Webanwendung umgesetzt werden soll, wird sich nach einem Vergleich dieser für eine Webanwendung entschieden.

Für die Entscheidung der Webkomponenten werden zwei Stacks miteinander verglichen. Dabei werden der FARM- und der MERN-Stack betrachtet. Am Ende wird der FARM-Stack gewählt, der aus den Komponenten *ReactJs*, *FastAPI* und *MongoDB* besteht.
Im Anschluss wird dieser Stack mittels eines „Proof of Concept“ realisiert. Dabei werden zusammenhängende Daten aus zwei *Collections* in einer Tabelle dargestellt. Dazu wird erläutert, wie *Collections* in *MongoDB* erstellt werden können. Zudem wird darauf eingegangen wie die Tabelle im Frontend erstellt wird. Dabei kann gezeigt werden, wie bestimmte Daten aus der Datenbank geholt und in der Weboberfläche angezeigt werden. Im Zuge dessen wird auch auf die Kommunikation von der Weboberfläche mit der Datenbank eingegangen.

Des weiteren wird erläutert, wie die QR-Codes mittels einer generierten Excel-Datei erzeugt werden. Da die Software eine bilinguale Ansicht bieten soll, wird auf die Umsetzung dieser Ansicht in der Weboberfläche eingegangen.

Zusammengefasst wurde in dieser Arbeit ein Konzept entwickelt um die *Datenmanagementsoftware* mittels einer Webanwendung zu realisieren.

7 Ausblick

In der Arbeit wurde das Konzept einer *Datenmanagementsoftware* erstellt und im Zuge eines „Proof of Concept“ wurden erste Bestandteile realisiert. Im Anschluss an diese Arbeit wird die *Datenmanagementsoftware* weiter realisiert.

Dazu kann in einem ersten Schritt die Datenbank um die bisher noch fehlenden *Collections* erweitert werden. Anschließend können die fehlenden, aber bereits konzipierten Oberflächen generiert und getestet werden. Mit diesen beiden Schritten kann die grundlegende Kommunikation zwischen den Benutzeroberflächen und der Datenbank finalisiert werden.

In den Anforderungen wurde das Qualitätsmanagementteam als Stakeholder identifiziert und bei der weiteren Konzeptionierung zwar berücksichtigt, jedoch wurden die spezifischen Ansichten im Rahmen der Arbeit nicht geplant. Daher müssen die Benutzeroberflächen für das Qualitätsmanagementteam designt und implementiert werden. Dazu muss geprüft werden, ob alle relevanten Daten für diese eingeplant wurden, um auch die ISO-Zertifizierung einhalten zu können.

Des Weiteren müssen alle Funktionalitäten, die an die Software gestellt wurden, umgesetzt werden. Um den gewünschten Anmeldeprozess zu ermöglichen, müssen die LDAP-Daten eingebunden und den Benutzern bestimmte Rechte zugeordnet werden. Zudem muss das Erzeugen der Excel-Datei für die Generierung der QR-Codes mit in die Software eingebunden werden. Außerdem muss das Scannen des Barcodes mit in die Software integriert werden und anschließend mit der Hardware getestet werden.

Wie im Ablauf der Kalibrierung beschrieben, muss den Mitarbeitenden mitgeteilt werden, wenn es Geräte gab, bei denen die Eingangsprüfung nicht erfolgreich war. Das wird momentan durch die Laborverantwortlichen kommuniziert, zukünftig soll dies jedoch durch ein automatisiertes Rundschreiben passieren. Das Rundschreiben enthält dann die Geräte, bei denen die Eingangsprüfung nicht erfolgreich war.

Bevor die *Datenmanagementsoftware* in den Produktivbetrieb eingebunden wird, ist eine ausgiebige Testphase nötig. Hierzu kann zunächst ein genauere Planung dieser Phase erfolgen. Während dieser Testphase sollte ein Hauptaugenmerk auf der Schnittstelle zwischen der Webanwendung und der Datenbank liegen.

Des Weiteren könnte man die beiden Teile des Gesamtprojekts miteinander verbinden. Dazu könnte die *Datenmanagementsoftware* als Oberfläche verwendet werden um ein neues Gerät mit den *SCPI*-Befehlen hinzuzufügen. Zusätzlich könnten über weitere Weboberflächen tabellarische Übersichten der Befehle hinzugefügt werden.

Quellenverzeichnis

[1] *Zentralinstitut für Engineering, Elekronik und Analytik (ZEA)*. abgerufen am: 20.06.2022. URL: https://www.fz-juelich.de/de/zea.

[2] *Electronic Systems (ZEA-2)*. abgerufen am: 20.06.2022. URL: https://www.fz-juelich.de/en/zea/zea-2/about-us.

[3] *ConSense Managementsystem*. abgerufen am: 08.07.2022. URL: https://www.consense-gmbh.de/home/.

[4] *Advantages of MongoDB over RDBM*. abgerufen am: 23.08.2022. URL: https://dev.to/meanii/why-use-mongodb-advantages-use-cases-39io.

[5] *MongoDB - Overview*. abgerufen am: 16.08.2022. URL: https://www.tutorialspoint.com/mongodb/mongodb_overview.htm.

[6] Marc Boeker. *MongoDB - Sag Ja zu NoSQL*. Entwickler.Press, 2010.

[7] *Schema Validation*. abgerufen am: 22.08.2022. URL: https://www.mongodb.com/docs/manual/core/schema-validation/.

[8] *HTML Styles - CSS*. abgerufen am: 23.08.2022. URL: https://www.w3schools.com/html/html_css.asp.

[9] *React Bootstrap*. abgerufen am: 23.08.2022. URL: https://react-bootstrap.github.io/.

[10] *MERN Stack Explained*. abgerufen am: 20.07.2022. URL: https://www.mongodb.com/mern-stack.

[11] *FastAPI*. abgerufen am: 23.08.2022. URL: https://fastapi.tiangolo.com/.

[12] *Schema Validation*. abgerufen am: 20.08.2022. URL: https://www.mongodb.com/docs/manual/core/schema-validation/.

[13] *React JSX*. abgerufen am: 16.08.2022. URL: https://www.w3schools.com/react/react_jsx.asp.

[14] *JavaScript Objects*. abgerufen am: 20.08.2022. URL: https://www.w3schools.com/js/js_objects.asp.

[15] *How To Use Axios With React*. abgerufen am: 15.08.2022. URL: https://www.freecodecamp.org/news/how-to-use-axios-with-react/.

[16] *ASGI Documentation*. abgerufen am: 19.08.2022. URL: https://asgi.readthedocs.io/en/latest/introduction.html.

[17] *CORS (Cross-Origin Resource Sharing)*. abgerufen am: 19.08.2022. URL: https://fastapi.tiangolo.com/tutorial/cors/.

[18] *Python Decorators Tutorial*. abgerufen am: 22.08.2022. URL: https://www.datacamp.com/tutorial/decorators-python.

[19] *Aggregation Pipeline*. abgerufen am: 19.08.2022. URL: https://www.mongodb.com/docs/manual/core/aggregation-pipeline/.

Jül-4436 • Januar 2023
ISSN 0944-2952

Mitglied der Helmholtz-Gemeinschaft